Verifiable Tally-Hiding Remote Electronic Voting

Von der Fakultät 5 (Informatik, Elektrotechnik und Informationstechnik)
der Universität Stuttgart zur Erlangung der Würde eines Doktors der
Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Julian Liedtke

aus Stuttgart

Hauptberichter:	Prof. Dr. Ralf Küsters
Mitberichter:	Prof. Dr. Véronique Cortier
Tag der mündlichen Prüfung:	02.02.2024

Institut für Informationssicherheit (SEC) der Universität Stuttgart

2023

Contents

List of Figures

List of Algorithms

List of Abbreviations

ACM	Association for Computing Machinery
AVS	Advanced Voting Solutions
CES	Civica Election Services
CPSNARK	commit-and-prove succinct non-interactive argument of knowledge
CRS	common reference string
DFG	Deutsche Forschungsgesellschaft
DPL	Debian Project Leader
EMV	electronic voting machine
ESC	Eurovision Song Contest
EUF-CMA	universal forgery under chosen-message attack
FHE	fully homomorphic encryption
GPPS	general-purpose proof system
IND-CCA2	indistinguishability under adaptive chosen ciphertext attack
IND-CPA	indistinguishability under chosen plaintext attack
IRV	instant-runoff voting
ITM	interactive turing machine
KTV	Küsters, Truderung, and Vogt
MMPR	mixed-member proportional representation
MP	Member of Parliament
MPC	multi-party computation
NIZKP	non-interactive zero-knowledge proof
SIAM	Society for Industrial and Applied Mathematics
SIG	Special Interest Group
SNARK	succinct non-interactive argument of knowledge
TEE	trusted execution environment
ZKP	zero-knowledge proof

Abstract

Electronic voting (e-voting) refers to casting and counting votes electronically, typically through computers or other digital interfaces. E-voting systems aim to make voting secure, efficient, convenient, and accessible. Modern e-voting systems are designed to keep the votes confidential and provide verifiability, i.e., everyone can check that the published election result corresponds to how voters intended to vote. Several verifiable e-voting systems have been proposed in the literature, with Helios being one of the most prominent ones.

However, almost all verifiable e-voting systems reveal not just the voting result but also the tally, consisting of the exact number of votes per candidate or even all single votes. Publishing the tally causes several issues. For example, in elections with only a few voters (e.g., boardroom or jury votings), exposing the tally prevents ballots from being anonymous, thus deterring voters from voting for their actual preference. Furthermore, attackers can exploit the tally for so-called Italian attacks that allow for easily coercing voters. Often, the voting result merely consists of a single winner or a ranking of candidates, so disclosing only this information, not the tally, is sufficient. Revealing the tally unnecessarily embarrasses defeated candidates and causes them a severe loss of reputation. For these reasons, there are several real-world elections where authorities do not publish the tally but only the result – while the current systems for this do not ensure verifiability. We call the property of disclosing the tally tally-hiding. Tally-hiding offers entirely new opportunities for voting. However, a secure e-voting system that combines tally-hiding and verifiability does not exist in the literature.

Therefore, this thesis presents the first provable secure e-voting systems that achieve both tally-hiding and verifiability. Our Ordinos framework achieves the strongest notion of tally-hiding: it only reveals the election result. Many real-world elections follow an alternative variant of tally-hiding: they reveal the tally to the voting authorities and only publish the election result to the public – so far without achieving verifiability. We, for the first time, formalize this concept and coin it *public tally-hiding*. We propose Kryvos, which is the first provable secure e-voting system that combines public tally-hiding and verifiability. Kryvos offers a new trade-off between privacy and efficiency that differs from all previous tally-hiding systems and allows for a radically new protocol design, resulting in a practical e-voting system. We implemented and benchmarked Ordinos and Kryvos, showing the practicability of our systems for real-world elections for significant numbers of candidates, complex voting methods, and result functions. Moreover, we extensively analyze the impact of tally-hiding on privacy compared to existing practices for various elections and show that applying tally-hiding improves privacy drastically.

Kurzzusammenfassung

E-Voting ist die elektronische Abgabe und Zählung von Stimmen über digitale Schnittstellen. Systeme für E-Voting sollen Wahlen sicher, effizient und zugänglich machen. Moderne E-Voting-Systeme schützen die Vertraulichkeit der Wählerstimmen und sind verifizierbar, d.h., man kann prüfen, dass die veröffentlichten Wahlergebnisse mit den tatsächlichen Stimmen übereinstimmen.

Bei den meisten Wahlsystemen wird nicht nur das endgültige Wahlergebnis veröffentlicht, sondern auch die genaue Anzahl der einzelnen Stimmen pro Kandidat oder sogar die Einzelstimmen. Die Veröffentlichung der Auszählung führt zu mehreren Problemen, wie mögliche Stigmatisierung oder Imageverlust von Wählern und Kandidaten. Insbesondere bei Wahlen mit einer geringen Anzahl von Wählern, wie Vorstandswahlen oder Juryabstimmungen, kann die Offenlegung der Gesamtzahl der Stimmen die Anonymität dieser beeinträchtigen und Wähler davon abhalten, ihre tatsächlichen Präferenzen zu wählen. Zudem kann die Veröffentlichung der genauen Stimmenauszählung zu einem schwerwiegenden Imageverlust für unterlegene Kandidaten führen. Deshalb veröffentlichen viele Wahlbehörden in realen Wahlen nur das Endergebnis, um die Vertraulichkeit zu wahren, allerdings auf Kosten der Verifizierbarkeit.

Die Eigenschaft des "Tally-Hiding" bezieht sich auf die Möglichkeit, lediglich das Endergebnis, wie den (oder die) Gewinner, und nicht die genaue Stimmenauszählung zu veröffentlichen. Dieser Ansatz eröffnet neue Möglichkeiten für elektronische Wahlen. Bisher gibt es in der Literatur jedoch kein sicheres E-Voting-System, das sowohl Tally-Hiding als auch Verifizierbarkeit sicherstellt.

Diese Dissertation stellt die ersten beweisbar sicheren E-Voting-Systeme vor, welche sowohl Tally-Hiding als auch Verifizierbarkeit gewährleisten. Das Ordinos-Framework gewährleistet die stärkste Form von Tally-Hiding, indem nur das Endergebnis offengelegt wird. Im Gegensatz dazu implementiert das Kryvos-Framework eine alternative Form des Tally-Hiding, die wir erstmals als "public tally-hiding" formalisieren. Es reflektiert die Praxis vieler realer Wahlen, bei denen nur das Ergebnis öffentlich bekannt gegeben wird, bei denen allerdings keine Verifizierbarkeit gewährleistet wird. Kryvos ist das erste beweisbar sichere E-Voting-System, das public tally-hiding und Verifizierbarkeit kombiniert. Dafür folgt Kryvos einem komplett neuen Protokolldesign. Wir haben sowohl Ordinos als auch Kryvos implementiert und Benchmarking unterzogen, um die Praktikabilität für reale Wahlen mit einer großen Anzahl von Kandidaten, komplexen Wahlmethoden und Ergebnisfunktionen aufzuzeigen. Diese Dissertation analysiert zudem ausführlich die Auswirkungen von Tally-Hiding in Bezug auf Vertraulichkeit und Anonymität im Vergleich zu bestehenden Verfahren für eine Vielzahl von Wahlen und zeigt, dass die Anwendung von Tally-Hiding die Privatsphäre deutlich verbessert.

1. Introduction

Voting is a fundamental democratic right that empowers people to shape their societies. By voting, we participate in the democratic process, ensuring our voices are heard and our opinions are represented. Every vote has the potential to make a difference, influence politics, and elect leaders who align with our values and aspirations.

We regularly conduct real-world elections for various purposes using different types of elections. These elections range from simple to highly complex ones. An election type consists of a choice space and an election result function. The choice space determines the possible ballots that the voters can cast, and the election result function determines how the election result is computed based on these ballots. Some of the most commonly used choice spaces and election result functions include:

Single-vote and *multi-vote* are the most basic choice spaces where voters can assign a single or multiple votes to a set of candidates. In its simplest form, we combine these choice spaces with the election result function that declares the candidate with the most votes as the election's winner or the ranked/unranked list of candidates with the most votes. This approach is commonly used in small to medium-sized elections in companies and associations to decide on one or more winners. Political elections also use single-/multi-vote but employ an additional election result function for computing the final seat distribution from the tally of all votes. Examples of such functions include the *Hare-Niemeyer method* (e.g., used in political elections in Luxembourg [Gal92] and Tunesia [fDI14]) and the *Sainte-Laguë* (e.g., used in political elections in Germany [Bun20], Indonesia [Okt22], Nepal [Kam22], and Norway [fDI23]). Another type of multi-vote choice space is *Borda voting*, where voters assign specific points among candidates according to a predetermined list of permitted point numbers. Borda is used to elect the winner of the Eurovision Song Contest (ESC) [Eur20] and national elections in the Republic of Nauru [Rep16].

A more involved election result function is called *majority judgment*, combined with a grading-based choice space where voters assign grades to each candidate instead of a specific number of votes. The winner is then the candidate with the best mean grade. This voting method is used, e.g., in political research polling in France, Germany, and the USA (see [BL10, BL14]).

Prominent examples of highly complex election result functions are the *Condorcet methods*, combined with ranking-based choice spaces where voters rank the candidates according to their preferences. The plain Condorcet criteria declares the candidate that would win against every other candidate in direct comparison as the winner. The Schulze method is a more complex Condorcet method used for internal elections by the Debian (Debian GNU/Linux) Project [Deb12]. Another prominent example of highly complex election result functions is

instant-runoff voting (IRV). IRV is also compatible with a ranking-based choice space. IRV computes the winner via an iterative process that mimics a multi-round election by progressively eliminating the candidate preferred by the least number of voters until a single winner remains. IRV is used, e.g., in political elections in Australia [NSW20], India [Gov20], the UK [The11], and the USA [Mai20].

The use of electronic voting systems provides many advantages over traditional paper-based elections. By using electronic voting systems, we can ensure greater accessibility and convenience for all voters, regardless of their physical location or mobility limitations. It allows for the seamless integration of modern technology, streamlining the voting process and reducing the likelihood of errors or discrepancies. Electronic voting can provide faster and more accurate results than paper-based elections, enabling a timely and efficient declaration of election results. Compared to paper-based elections, e-voting offers enhanced security controls, such as encryption and authentication, that help safeguard the security of the voting process. Furthermore, electronic elections can drastically reduce the carbon footprint compared to their paper-based counterparts [WK23].

There are two primary methods of electronic voting:

1. Physical e-voting involves electronic voting machines placed in polling stations and supervised by representatives of governmental or independent electoral authorities. Such elections have been performed, e.g., in Germany [Bun09], India [Rao10], the Netherlands [Hae08] on a federal level, and in the following states in the USA: California [Bow07], Florida [ctV23], Mississippi [Vot07], Pennsylvania [Com07], and Virginia [Vir15].

2. Remote e-voting, the focus of this thesis, allows voters to submit their votes electronically to the election authorities from any location via the Internet using their personal devices. Compared to voting machines at polling stations, the Internet does not provide secure communication channels, making security guarantees even more challenging. Such elections have been performed, e.g., in Australia [scy17a], Estonia [DM04], and the Swiss Confederation [Bow07] on a federal level, but also widely in companies, organizations, and associations (see, e.g., [ACM20, Cro19, Deu19, Soc19]).

When using e-voting systems for elections, these systems must meet specific security requirements, which are as follows:

- *Verifiability* (see [CGK$^+$16]): The essential requirement of any election is verifiability, which we will explain below.
- *Accountability* (see [KTV10]): If we detect that the election result is incorrect, it becomes necessary to identify and hold any parties accountable that may have misbehaved. Thus, accountability motivates parties not to temper the election since misbehaving parties will be held responsible.

- *(Vote-)Privacy* (see [BCG+15b]): When it comes to electronic voting systems, it is essential to maintain individual votes confidential to ensure the voting process's secrecy. To measure the advantage that an attacker can gain in accessing the plain votes of voters, we use the concept of δ-privacy (see Section 2.5 and [KTV11]).

Another crucial feature of e-voting systems is *feasibility/practicalibility*. We must ensure that the e-voting scheme is efficient enough to allow voters to create and cast their ballots promptly, without delays and long waiting times. In addition, the system shall be designed in such a way that it can tally the election results accurately within a reasonable timeframe. This requires thorough consideration of the technical capabilities of the system, as well as the size and complexity of the election being conducted.

1.1. Verifiabiliy

The essential requirement of an election is verifiability. Verifiability states that we can detect and identify any issues or discrepancies, allowing for a fair and just outcome. E-voting systems have faced significant issues worldwide, such as undetected loss or incorrect counting of votes. In the following, we present real-world examples of elections that suffered from inadequate security, including two elections that used voting machines and two remote elections.

1.1.1. Insecurity of Electronic Voting Machines employed in India

Since their introduction in the 1990s, the so-called *electronic voting machines (EVMs)* [ASI06] are used extensively in India for conducting various types of elections, including local government elections, national parliamentary elections, state legislative assembly elections, municipal elections, Panchayat (various local governments) elections, and legislative council elections [Rao10]. EVMs are a significant part of India's election infrastructure and have streamlined the voting process in the country. However, there have been debates and discussions about their security and transparency, with some concerns raised by various groups about the trustworthiness of EVMs and the need for paper trails or other forms of verification.

The study in [WWH+10] focuses on understanding the weaknesses and potential risks associated with EVMs employed in India. The authors present their findings by analyzing a randomly selected EMV. They demonstrate a successful attack on the EMV's security by replacing the microcontroller with a modified version. The manipulated EMV can alter the voting results without being detected. The paper emphasizes that the security controls implemented in the EMV are inadequate to prevent such attacks. Therefore, these elections do not provide integrity and thus possess no verifiability guarantees [WWH+10].

1.1.2. Insecurity of Electronic Elections conducted in Estonia

In 2003, Estonia made history by being the first country to offer remote online voting for a parliamentary election [Ins23, DM04]. Before, the system was tested in local elections earlier

that year. Since 2005, e-voting has also been used in local elections, marking the second time online voting was employed in Estonia. Additionally, e-voting has been successfully implemented in European Parliament elections since 2007 in Estonia. See [Ins23] for an overview. However, in 2013, researchers expressed concerns about the trustworthiness of Estonia's e-voting system, highlighting vulnerabilities and potential risks that break verifiability.

The study [SFD+14] highlights the feasibility of an attacker at the state level, a sophisticated criminal, or a dishonest insider manipulating election results by bypassing technical and organizational controls. Even if the attacker does not go that far, there are several ways in which they could interfere with the voting process or cast doubt on the correctness of the results. The election system in question featured an insecure build system and insecure software. Furthermore, it used peripheral devices like personal USB sticks, for which no security guarantees were given. Moreover, official videos of the October 2013 municipal elections show Wi-Fi credentials, national ID PINs, and keystrokes of the root password, which can be used to alter the election results without being detected.

In summary, this study shows that there are no guarantees of verifiability for Estonia's election in 2013.

1.1.3. Insecurity of WinVote Machines employed in the USA

Advanced Voting Solutions (AVS) [Uni19] was a company that sold WinVote voting machines [Ver23]. These machines were utilized during elections in Mississippi [Vot07], Pennsylvania [Com07], and Virginia [Vir15]. The AVS WinVote machine's architecture included a touchscreen that ran on Windows XP. It also had a thermal paper printer and a USB port behind a lock. Unfortunately, as shown in [Eps15], the system had several weaknesses. For instance, for the elections in 2014 held in Virginia, the AVS machines use an outdated version of Windows XP that has not received any patches since 2004. Additionally, it had weak wireless security with hardcoded passwords, no option to turn off wireless, and lax physical security.

It was shown in [Eps15] that election results can be tampered with using a laptop and free tools without being detected. An attacker could intercept network traffic, gain access to the voting machine through an easily obtainable admin password, manipulate the Microsoft Access database that holds the votes, and then upload the modified database back onto the machine. Even without factoring in any software or hardware vulnerabilities, the AVS WinVote system has been demonstrated to be fundamentally insecure.

1.1.4. Insecurity of Electronic Elections in Switzerland and Australia

Scytl's voting system, sVote [scy23b], has been certified and used in four Swiss cantons since 2016 [scy17b]. It is also utilized for Swiss referendums and federal elections in 2019 [Tea19], as well as state elections in New South Wales, Australia [scy17a].

According to the analysis in [HLPT20], sVote version 2.1 has severe issues with its underlying cryptographic primitives. It uses these primitives in settings where they are not secure and

employs cryptographic protocols that do not provide the required level of security. Additionally, the system lacks evidence of how specific values are generated, allowing malicious parties to cheat by exploiting potential trapdoors. As a result, using sVote version 2.1, it is possible to change the election outcome without being detected.

1.1.5. Achieving Verifiability

As these examples show, achieving verifiability is not trivial. E-voting systems are very complex, involving several hardware and software components. Due to the high complexity of these systems, identifying, assessing, and addressing vulnerabilities is not easy. Even minor implementation errors and security flaws can significantly affect the voting process's accuracy and integrity. Therefore, it is essential to have thorough validation and verification protocols in place to ensure that e-voting systems are secure, reliable, and trustworthy.

Verifiability is the fundamental security property not just necessary in traditional paper-based elections, where multiple entities and observers can monitor the tallying process, but also in electronic voting systems. Verifiability is essential in detecting any irregularities that may arise during the election process, such as voter suppression, manipulation of voting machines, or miscounting ballots.

Verifiability should be possible even when the voting devices or servers have programming errors or are malicious. It helps maintain transparency, democracy, public trust, and accountability. It is the foundation of the electoral process's integrity and legitimacy, contributing to the stability and proper functioning of democratic societies.

To achieve verifiability, the majority of current e-voting systems adopt a method similar to traditional paper-based elections. They calculate and then publish the aggregated tally, which contains the number of votes each candidate received or individual votes. Additionally, they provide supplementary information showing the aggregated tally's correctness. With this information, anyone can independently compute the election result using a specific election result function, such as determining the winning candidate (the one that received the most votes) or identifying the top n candidates. Assuming that we can detect any errors occurring to obtain the aggregated tally (e.g., voters can check whether their intended votes are part of the aggregated tally), this approach leads to a verifiable system since computing the election result is solely based on verified public information.

Although this approach is commonly employed and may be deemed necessary in the absence of alternative methods, disclosing the aggregated tally as an intermediate value for determining the election result carries various disadvantages:

- *Biased voters:* When elections are carried out over multiple rounds, it is vital to consider the potential ramifications of disclosing intermediate tallies of previous rounds. Publicizing these tallies may inadvertently introduce bias or sway voters' decisions for subsequent rounds of voting. This poses a significant problem if voters perceive their preferred candidate as unlikely to win based on the current tallies and consequently alter their vote in a manner that

does not align with their true preferences, altering the election result. By maintaining the confidentiality of intermediate tallies, the electoral process upholds fairness and impartiality and enables all candidates to compete equitably.

- *Embarrassed candidates:* When considering the publication of election results, it is crucial to acknowledge the potential impacts. In particular settings, such as businesses or associations, disclosing the exact number of votes the candidates received may be unwise. This is due to causing a loss of reputation or discomfort for the losing candidates, who may feel exposed or vulnerable. Additionally, if the margin of victory is narrow, releasing the aggregated tally undermines the authority of the elected individual and results in a weak mandate. These outcomes might be undesirable for any organization or group. Therefore, carefully considering the implications of publishing election results is necessary, weighing the potential advantages against the potential drawbacks.
- *Gerrymandering:* Gerrymandering refers to intentionally altering the boundaries of electoral districts to gain an advantage for a particular political party or group. In other words, it is a deliberate manipulation of voting district boundaries aimed at maximizing electoral gains or minimizing the influence of rival parties. Gerrymandering can be facilitated by the public release of voting data, which can then be used to redraw district boundaries to favor one party over the other. Gerrymandering happens world-wide, for example, in Australia [Sto33], India [Ind19, TRT22], Hungary [Eco22], and in California, USA [Pro11].
- *Confidentiality of votes:* The confidentiality of votes is fundamental to democratic elections. If the tally is made public or exposed, it could harm the election results. This is because voters may be hesitant to vote for their actual preference, fearing their choice will be known and possibly used against them. The secrecy of voting is essential for ensuring that voters can express their choices without fear of repercussions. By protecting the confidentiality of the votes, we can help to ensure that democratic elections are free, fair, and democratic.
- *Italian attacks:* Italian attacks arise from the issues mentioned in the point above. In preferential voting systems such as instant-runoff voting (IRV), each voter's choice can be a unique identifier even when only a few candidates exist. This means that the privacy of each voter is diminished, which adversaries can exploit through a technique called *Italian attacks* [RCPT19]. With access to the aggregated tally, the adversary can examine the data and coerce any voter to vote for an unlikely option. The adversary can then verify whether or not the voter followed the coercion. Such attacks seriously threaten the integrity of the voting process and must be considered when implementing preferential voting systems.

These issues arise from publishing the aggregated tally, which reveals the number of votes each candidate receives or individual votes. To mitigate these issues, some elections have put restrictions in place to prevent the disclosure of the aggregated tally. An example is the elections for the Special Interest Groups (SIGs) in the Association for Computing Machinery (ACM) [ACM20]. These elections announce only the winner and do not publish the aggregated

tally. Similarly, the Gremienwahl of the Deutsche Forschungsgesellschaft (DFG) keeps the number of votes received by the losing candidates confidential [Deu19]. Civica Election Services (CES), a renowned e-voting provider, conducts several dozen elections annually where customers request only the actual election result from CES. According to CES, this approach is implemented to minimize the possibility of weak mandates or gerrymandering issues CES. Further examples are the elections carried out in CrossRef [Cro19] and in Society for Industrial and Applied Mathematics (SIAM) [Soc19]. However, since these elections only publish the results, external observers cannot verify the claimed results. These real-world elections are not verifiable, and so far, no method is known to achieve verifiability in such voting schemes.

1.2. Tally-Hiding

Following [SP15], we refer to e-voting systems that only disclose the election result as *tally-hiding* systems. Previous proposals for tally-hiding e-voting systems, such as those by Benaloh [Ben86], Hevia and Kiwi [HK04], and Wen and Buckland [WB09], were limited in their applicability to specific voting methods and lacked formal proofs of security, making them impractical solutions.

We can divide tally-hiding into three categories:

1. *Full tally-hiding.* Fully tally-hiding e-voting systems offer the highest level of privacy for the tally and ballots. These systems guarantee that no one, not even external observers, voters, election officials, or trustees, can access any information regarding the election except for the final result. Full tally-hiding ensures that the actual tally is kept hidden, and any interim values used to calculate the election outcome are inaccessible to all parties. The work of [SP15] formalized this notion for the first time.

2. *Partial tally-hiding.* While fully tally-hiding e-voting systems conceal all information beyond the final election result, research has proposed e-voting systems that only hide specific parts of the tally in order to address particular issues. Partially tally-hiding systems are designed to hide only certain aspects of the tally and thus reveal some information besides the election's result. For example, consider a multi-round runoff election between three candidates, A, B, and C, where the candidate with the least votes is eliminated in each round. A fully tally-hiding voting scheme reveals the winner of the election, e.g., candidate B. In comparison, a partially tally-hiding voting scheme could reveal which candidate is eliminated in each round, e.g., the election result (C, A) denotes that candidate C is eliminated in the first round and candidate A in the second round, leading to candidate B as the winner of the election.

3. *Public tally-hiding.* As explained above, in many real-world elections, the voting authorities compute the election result internally by learning the aggregated tally and then only publish the final result. This concept differs from full and partial tally-hiding in that it essentially

ensures full tally-hiding towards the public but reveals the aggregated tally towards some designated parties running the election.

As the abovementioned real-world examples demonstrate, constructing an e-voting system that only achieves tally-hiding is not difficult. These voting systems only publish the election results. However, the real challenge is integrating tally-hiding and verifiability to enable observers to confirm the outcome without revealing any extra information about the tally. Unfortunately, a provably secure e-voting system combining verifiability and tally-hiding does so far not exist in the literature. Our work of [KLM+20a] is the first to propose a secure verifiable and tally-hiding e-voting system, called Ordinos. Ordinos is a generic framework that can be instantiated for choice spaces and election result functions. In the work of [KLM+20a], we lay the foundation to verifiable fully tally-hiding e-voting by instantiating the Ordinos framework for relatively simple single- and multi-vote elections. It remains an open question whether and how efficiently we can conduct more complex elections in a verifiable and tally-hiding manner.

1.3. Goal and Contributions of This Thesis

There are four major goals for this thesis:

1. We want to instantiate the Ordinos framework to construct verifiable and fully tally-hiding e-voting systems covering the Hare-Niemeyer and Sainte-Laguë methods, several Condorcet methods, and instant-runoff voting.
2. We want to explore the impact of tally-hiding regarding the secrecy of the tally and confidentiality of individual votes.
3. We want to formalize the concept of public tally-hiding.
4. We want to construct the first secure e-voting system that provides verifiability *and* public tally-hiding following the concept of existing public tally-hiding practices. We want to construct such systems supporting various elections. This includes various choice spaces like single- and multi-vote, Borda, ranked- and grading-based voting methods, and several election result functions like Condorcet methods, majority judgment, and instant-runoff voting.

In what follows, we present our contributions categorized into the three types of tally-hiding.

Part I – Full Tally-Hiding

First, we explore the possibilities of constructing a secure, verifiable, and fully tally-hiding e-voting system. Advanced cryptographic techniques such as fully homomorphic encryption (FHE) and universally verifiable and accountable multi-party computation (MPC) are required

to achieve this level of privacy. These techniques enable the trustees to compute the election result without accessing any information that could compromise the confidentiality of ballots and the tally.

Our work of [KLM+20a] is the first to propose a secure verifiable and tally-hiding e-voting system, the Ordinos framework, which we instantiated for relatively simple single- and multi-vote elections.

Our Contributions. Our work in Chapter 3 instantiates the Ordinos framework to securely and efficiently support a variety of real-world elections.

Ordinos is the first provably secure, verifiable, and fully tally-hiding remote e-voting protocol, achieving verifiability and accountability. Unlike previous protocols, Ordinos is a *modular framework* for fully tally-hiding e-voting that can be instantiated for supporting a wide range of election types, including essentially arbitrary voting methods. Our security analysis and proof are performed generically for the Ordinos framework itself and, therefore, carry over to all such instantiations. Creating an instantiation of Ordinos mainly boils down to constructing a universally verifiable/accountable MPC component for computing the result of the desired voting method. It is challenging to develop an MPC protocol that enjoys all necessary security properties, is sufficiently efficient in practice, and is compatible with the rest of the system.

We propose and implement various instantiations of Ordinos for commonly used single- and multi-vote elections. We also include several other complex voting methods like the Hare-Niemeyer and Sainte-Laguë methods, several Condorcet methods, and instant-runoff voting. This requires designing multiple new universally verifiable MPC protocols. We make appropriate design decisions to ensure practical performance in everyday, real-world settings. Our extensive evaluations demonstrate that these instantiations achieve practical performance in everyday real-world settings.

We design an Ordinos instantiation that performs the entire election process for the German federal parliament (Bundestag) in a verifiable and fully tally-hiding manner. The German federal election is one of the most complicated real-world elections, with millions of voters, dozens of parties, hundreds of individual candidates, and hundreds of electoral constituencies. The election process involves using sophisticated multi-step algorithms to compute the election result, which includes assigning seats to individual candidates. The process involves combining the results of different constituencies, distributing seats that are directly allocated to individual candidates instead of parties, taking into account minimum vote counts for parties before they receive any seats, including particular exceptions for minorities, possibly increasing the size of the parliament, and assigning party seats to individual candidates according to party candidate lists for each constituency and weighted by how many votes a party has obtained in the respective constituency. We have tested our system on the data from the 2021 election and demonstrated that we can perform even such a complex election on a real-world scale in a fully tally-hiding manner. This powerful insight highlights the potential for using Ordinos in significant and complex real-world elections.

Finally, we provide *a web framework of* Ordinos *that supports all steps of the voting process*, including setting up elections, the vote submission (including voter authentication), the secure tally-hiding evaluation of the election result, and all verification steps necessary to verify the result of the election. The web framework fully supports all our Ordinos instantiations and voting methods, except for the German Bundestag elections.

Moreover, we provide insights into the impact of tally-hiding by analyzing and comparing privacy in different elections with and without tally-hiding. Our analysis shows that tally-hiding improves privacy severely.

Part II – Partial Tally-Hiding

In contrast to fully tally-hiding e-voting systems, which aim to hide all information beyond the final election result, research proposed several partial tally-hiding e-voting systems that attempt to hide only specific parts of the tally (from everyone, including trustees) to solve certain specific issues, most notably Italian attacks (e.g., [RCPT19] for IRV and [BMN+09, Hea07] for single transferable vote, see Section 2.7). Such systems can be more efficient than fully tally-hiding systems since not all intermediate information (e.g., the ranking of losing candidates) must be kept secret. Hence, partial tally-hiding is a relaxation of full tally-hiding to improve efficiency.

Our Contributions. In Chapter 4, we extend the privacy analysis of voting systems by computing privacy values for partial tally-hiding election result functions and comparing them with their tally-hiding and non-tally-hiding counterparts.

We constructed the Ordinos framework for full tally-hiding, but we can apply it for partial tally-hiding by using suitable election result functions. For example, considering the multi-round runoff election introduced above, we can construct a fully tally-hiding instantiation of the Ordinos framework that computes the eliminated candidate per round. We can then use this fully tally-hiding instantiation for the elimination result function to compute the election result function that outputs just the winner in a tally-hiding manner.

Applying the Ordinos framework, we propose several partial tally-hiding Ordinos instantiations and analyze their trade-off between privacy and efficiency in comparison to their corresponding full tally-hiding Ordinos instantiations.

Combining the privacy analysis and the partially tally-hiding Ordinos instantiations, we show that by employing particular partial tally-hiding functions, we can achieve privacy levels similar to the full tally-hiding counterparts while significantly improving the performance of the election.

Part III – Public Tally-Hiding

The concept of public tally-hiding is motivated by existing practices: As presented above, there are many cases where voting authorities chose to compute internally but not publish the aggregated tally; instead, they revealed only the election result to avoid issues regarding privacy. Hence, while the trustees learn the aggregated tally, the public only learns the election

result. While publicly tally-hiding elections are standard, they currently do not offer verifiability: Neither voters nor external observers can confirm the correctness of the election results.

The concept of public tally-hiding is further motivated by allowing for more efficient design choices compared to full tally-hiding. Fully tally-hiding e-voting systems require heavy-weight cryptography, such as FHE and MPC, to conceal the tally and all intermediate results, even from the trustees. Consequently, these systems are computationally expensive and may not scale well for highly complex voting methods or the desired number of candidates. Allowing the trustees to learn the aggregated tally might allow for more efficient solutions.

Our Contributions. In Chapter 5, we propose and study *publicly tally-hiding* techniques. These differ from all previous tally-hiding approaches and offer a new trade-off between privacy and efficiency. More specifically, we contribute the following:

- Since the concept of (verifiable) publicly tally-hiding elections is novel and has not been investigated so far, this thesis develops a *general definitional security model for publicly tally-hiding e-voting*, thereby also providing the first formal definition of the public tally-hiding property. With this, we enable the rigorous and formal modeling of publicly tally-hiding e-voting systems and their security properties, including vote privacy and verifiability.
- We analyzed privacy for public tally-hiding. We show that the privacy results regarding full tally-hiding also apply to public tally-hiding regarding the public. Moreover, we extend our results regarding privacy regarding full tally-hiding by analyzing further and more complex voting methods.
- Another main contribution of this thesis is the Kryvos framework, which is the *first provably secure verifiable publicly tally-hiding e-voting system.* The Kryvos framework is a flexible voting scheme that can be customized for different types of elections by designing or fixing appropriate cryptographic building blocks, just like Ordinos. The framework uses general-purpose proof systems (GPPSs, see, e.g., [Set20]) to ensure the verifiability of the final result without revealing the tally. It is a versatile system that does not require a specific GPPS but can work with any standard instantiation, including succinct non-interactive arguments of knowledge (SNARKs, see, e.g., [GMO16, Gro16, AHIV17, BBC$^+$18, BBHR18, BBB$^+$18, BCR$^+$19, ESLL19, ESS$^+$19, GWC19, MBKM19, AC20, ALS20, BFH$^+$20, BLNS20, COS20, ENS20, Set20, ACK21, AL21, BCS21, ISW21, LNPS21, ACC$^+$22, ACL$^+$22, CHJ$^+$22, Eag22, LNP22, BS23, ABST23, CBBZ23, CGG$^+$23, CGI$^+$23, GLS$^+$23, Zha23]). To implement Kryvos for a specific voting method, we need to propose an instance of the GPPS that proves the correct computation of the result of that voting method.
- To carry out elections using Kryvos, we develop and implement optimized versions of the state-of-the-art Groth16 SNARK [Gro16] for a wide range of elections, including single- and multi-vote, Borda, ranking and grading choice spaces, as well as Condorcet methods, IRV, and majority judgment. With these Groth16 instances, we have created instantiations of Kryvos for many commonly used voting methods, from simple to complex. Via thorough

performance testing, conforming our hypothesis, we demonstrate that our publicly tally-hiding Kryvos framework performs significantly better than Ordinos instances. In particular, for highly complex voting methods such as Condorcet and IRV, our Kryvos instances support more candidates than Ordinos instances. Moreover, our Kryvos scheme outperforms even the state-of-the-art IRV partial tally-hiding e-voting scheme of [RCPT19], while achieving incompatible privacy properties (see Section 5.4).

- As a significant byproduct, we provide highly efficient zero-knowledge proofs (ZKPs) for showing ballot validity for several choice spaces. We do not only construct ZKPs for relatively simple choice spaces like single- and multi-vote, but most impactfully for very complex choice spaces, for which no ZKPs exist so far. For instance, we are the first to propose ZKPs for the choice space of Borda tournament style (see Section 2.6). Our constructions offer high performance, allowing voters to prove the validity of their ballots very efficiently. These outcomes are not limited to public tally-hiding. They can be used in other e-voting schemes, providing a new foundation to handle choice spaces.

Publications

This thesis is based on five publications published in international security conferences. The underlying ideas have been developed during joint discussions and meetings with the co-authors of the publications listed below. We list the contributions of Julian Liedtke in detail below.

- Ordinos*: A Verifiable Tally-Hiding E-Voting System (EuroS&P 2020) [KLM+20a] (with a technical report [KLM+20b]).* This work lays the foundation of the Ordinos framework, including a proof-of-concept implementation of plurality voting (and slight variants thereof) and the security proofs. The proof of verifiability/accountability and the proof of privacy were done by Johannes Müller and are part of his PhD thesis [Mül19]. Andreas Vogt did the proof of the ideal privacy level. Andreas Vogt also computed the privacy values provided in [KLM+20a, KLM+20b] (Figures 3.2 to 3.5 in this thesis). Julian Liedtke constructed the instantiations provided in this work and implemented them based on an initial proof-of-concept implementation by Johannes Müller. Chapter 3 in this thesis, in particular the description of the Ordinos framework in Section 3.2.1, covers the content and is based on this publication and the corresponding technical report, with some parts taken verbatim from [KLM+20a, KLM+20b].
- *Extending the Tally-Hiding* Ordinos *System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting (E-Vote-ID 2021) [HHK+21a] (with a technical report [HHK+21b]).* This work instantiates the Ordinos framework for several (real-world) elections. Fabian Hertel and Jonas Kittelberger supported Julian Liedtke in implementing and creating benchmarks. Chapter 3 in this thesis, in particular various instantiations of the Ordinos framework presented in Section 3.2.3, covers the content and is based on this publication

and the corresponding technical report, with some parts taken verbatim from [HHK+21a, HHK+21b].

- *Fully Tally-Hiding Verifiable E-Voting for Real-World Elections with Seat-Allocations (ESORICS 2023) [WLH+23a] (with technical report [WLH+23b]).* This work studies seat-allocation methods and constructs further instantiations of the Ordinos framework for such voting schemes. Julian Liedtke modified the Ordinos framework for elections using multiple constituencies and has done the formal proofs. Carmen Wabartha (under the guidance of Julian Liedtke) and Julian Liedtke jointly constructed and implemented the instantiation for the German Bundestag. Chapter 3, in particular the extension of the Ordinos framework for multiple constituencies presented in Section 3.2.4, covers the content and is based on this publication and the corresponding technical report, with some parts taken verbatim from [WLH+23a, WLH+23b].
- Ordinos*: Remote Verifiable Tally-Hiding E-Voting - A Fully-Fledged Web-Based Implementation (E-Vote-ID 2023) [LAA+23b].* This work provides a full-fledged web-based implementation of the Ordinos framework. Several students supported Julian Liedtke in implementing the system. Section 3.2.5 in this thesis covers the content and is based on this publication, with some parts taken verbatim from [LAA+23b].
- Kryvos*: Publicly Tally-Hiding Verifiable E-Voting [HKK+22a] (with a technical report [HKK+22b]).* This work contains the Kryvos framework. Johannes Müller supported Julian Liedtke in proving security. Andreas Vogt computed privacy values for IRV (Figure 3.7 in this thesis). Chapter 5 and appendices B.1 and B.2 in this thesis cover the content and are based on this publication and the corresponding technical report, with some parts taken verbatim from [HKK+22a, HKK+22b].

In addition to the privacy values computed by Andreas Vogt in the above publications, Julian Liedtke has computed further privacy values as part of writing this thesis. The privacy values of Figures 3.2 to 3.5 and 3.7 in this thesis are computed by Andreas Vogt, all other privacy values are computed by Julian Liedtke.

In addition to the above works, Julian Liedtke has also collaborated on the work of [RLH+23], which is not part of this thesis.

1.4. Structure of This Thesis

We lay the foundation of electronic voting (e-voting) in Chapter 2. That is, we present a typical (e-)voting flow and then define security notions in the context of e-voting. Furthermore, this section presents vital components of every election, namely various choice spaces and election result functions. Furthermore, we describe real-world elections.

In the following three chapters, we present the different types of tally-hiding. In Chapter 3, we present full tally-hiding and Ordinos, the first provable secure fully tally-hiding e-voting

system. In Chapter 4, we present partial tally-hiding and variants of Ordinos that fulfill this property. In Chapter 5, we present public tally-hiding and Kryvos, the first provable secure publicly tally-hiding e-voting system. We conclude in Chapter 6.

2. Electronic Voting

Elections are crucial for democracy. Politics, companies, organizations, associations, political parties, non-profit, and religious institutions regularly held elections (see, e.g., [BL10, fCR10, Rao10, BL14, Cro19, Deu19, KTV+19, Soc19, ACM20, deb22, Ins23, Pol23, Scy23a]). Different voting methods and election result functions exist to determine the election winner. Voting methods range from simple, like plurality/single-choice, to complex, like cumulative and preferential/multi-round voting. In practice, elections commonly utilize simple voting methods, but more complicated ones can better reflect the voters' preferences. For instance, instant-runoff voting (IRV) is a preferential voting method many countries use for municipal or national political elections, including Australia [NSW20], India [Gov20], the UK [The11], and the US [Mai20]. We define the valid votes/choices a voter can cast as a choice space $\mathfrak{C}$. Different result functions are used in elections, such as determining the winner (e.g., in presidential elections), selecting the n best or worst candidates (ranked or not ranked) for positions, or deciding who advances to a runoff election. We denote election result functions with f^{res}.

There is rising interest in performing this wide variety of elections via electronic voting (e-voting) systems, particularly remote e-voting via the Internet. More and more companies, organizations, associations, and even political bodies have adopted remote e-voting systems for their elections (see, e.g., [ACM20]). Internet voting has been employed in numerous official and binding elections across various government levels worldwide. For example, over two million remote Internet voting opportunities have been in over 90 local Canadian elections [Goo14].

This demand has further increased due to the recent COVID-19 pandemic [HGM+22]. In order to meet the increasing demand for e-voting solutions, many IT companies offer their services [Hei17], including, for example, Microsoft [Bur19]. Several e-voting systems have already been deployed in binding elections (see, e.g., [Adi08, CCC+08, AdMPQ09, BCH+12, CRST15, GGP15]), with Helios [Adi08] being one of the most prominent (remote) e-voting systems used in practice.

Two main types of e-voting systems are currently in use. The first type requires the voter to physically go to a polling station and cast their vote using a voting machine. The second type, which is the main focus of this thesis, allows the voter to vote remotely over the Internet using their device. This type of e-voting system is becoming increasingly popular due to its convenience and accessibility.

The structure of this chapter is as follows. In Section 2.1, we present the typical (e-)voting flow. In order to formally define the security of e-voting systems, we present the general computational model for capturing e-voting systems in Section 2.2. We formally define verifiability in Section 2.3.

Then, in Section 2.4 we define accountability, and finally, we define the privacy of e-voting systems in Section 2.5. We then present various commonly used choice spaces (Section 2.6) and election result functions (Section 2.7). In Section 2.8, we describe and explain real-world elections and show which choice spaces and election result functions they use. Finally, in Section 2.9, we briefly present existing e-voting systems that we will discuss later at suitable places in this thesis.

2.1. Voting Flow

The Helios [Adi08] remote e-voting system is commonly used as a framework for other e-voting schemes. Our e-voting schemes in this thesis also apply the Helios framework. As a result, this section will outline a typical election procedure employing Helios.

The following participants typically participate in a Helios-like e-voting protocol: a voting authority Auth, (human) voters $\mathsf{v}_1, \ldots, \mathsf{v}_{n_{\mathsf{voters}}}$, their voter-supporting devices $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$, trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$ (also called talliers), an authentication server AS, and an append-only bulletin board BB.

A vital building block of Helios is a homomorphic, IND-CPA-secure $(t, n_{\mathsf{trustees}})$-*threshold public-key encryption scheme* $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, E_{\mathsf{pk}}, \mathsf{dec}^{\mathsf{share}}, \mathsf{dec})$ (formally defined in Appendix A.1), e.g., the exponential ElGamal or Paillier encryption scheme. The shares of the secret key are distributed among the trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$ such that at least t trustees need to cooperate in order to decrypt a ciphertext. Furthermore, the homomorphic property of the encryption schemes allows to operate on the ciphertexts that add the corresponding plaintexts, i.e., the operation $E_{\mathsf{pk}}(a) \oplus E_{\mathsf{pk}}(b)$ creates a ciphertext with plaintext value $a + b$, where pk denotes the public key. Likewise, we write $\bigoplus_{i=1}^{n} E_{\mathsf{pk}}(a_i)$ to denote the operation that creates a ciphertext with plaintext value $a_1 + \cdots + a_n$.

At its core, electronic voting operates as a decentralized system involving the participants stated above. The voter v_i utilizes her voter-supporting device (VSD_i), such as a desktop computer or smartphone, connected to the voting system via a web browser or a dedicated application, to generate a ballot representing her choice $c_i^{\mathsf{choice}} \in \mathfrak{C}$. Such choices can be simple (e.g., voting for one candidate) or complex (e.g., ranking a selected subset of candidates). All possible choices a voter can select are captured in the *choice space* $\mathfrak{C}$. The choice space determines the valid choices for which voters can cast ballots. For this, the parameters $n_{\mathsf{components}} \in \mathbb{N}$ denote the number of *(choice) components* of an election and a set of valid voting choices $\mathfrak{C} \subseteq \mathbb{N}^{n_{\mathsf{components}}}$. Both parameters, $n_{\mathsf{components}}$ and $\mathfrak{C}$, are instantiated depending on the voting method.

We present the following two phases (the voting and the tallying phase) as a sequence diagram in Figure 2.1.

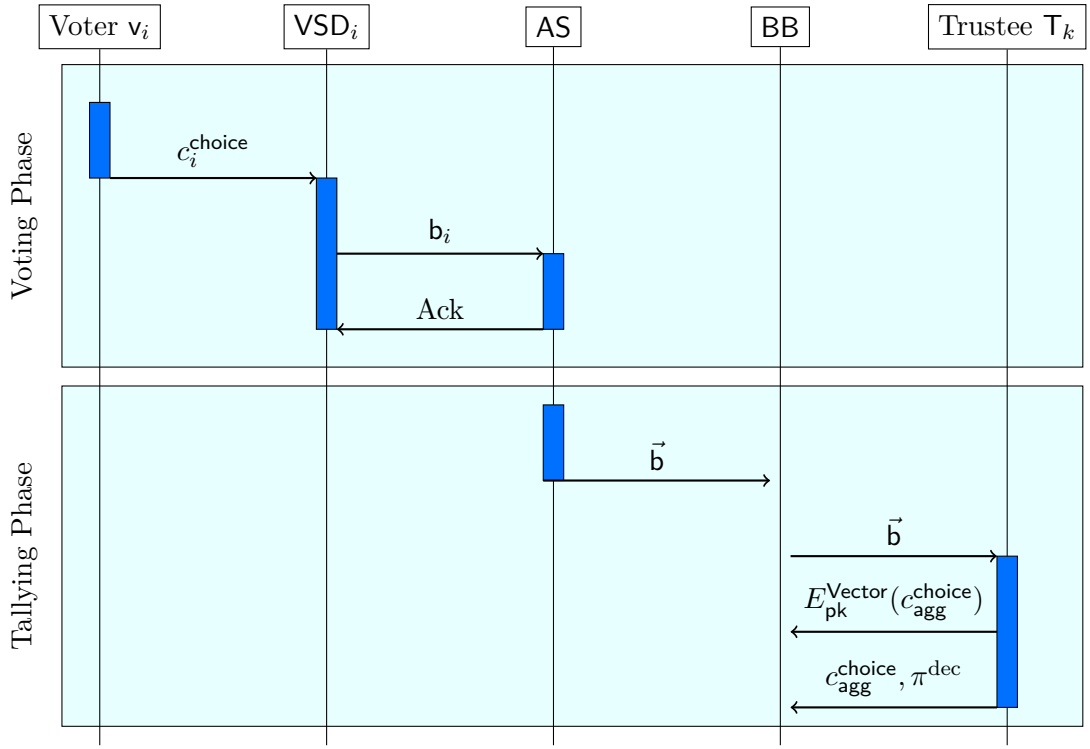


Figure 2.1.: Sequence diagram of the voting and tallying phases.

2.1.1. Voting Phase

A voter v_i selects her choice $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C}$. For example, in an election between n_{cand} candidates, where the voters are allowed to vote for exactly one candidate, the choice space is defined as $\{(c_i^{\mathsf{component}})_{i=1}^{n_{\mathsf{components}}} \mid n_{\mathsf{components}} = n_{\mathsf{cand}}, c_i^{\mathsf{component}} \in \{0,1\}, \sum_{i=1}^{n_{\mathsf{components}}} c_i^{\mathsf{component}} = 1\}$. In this case, a (choice) component represents a candidate. We present various choice spaces in Section 2.6. In summary, every voter v_i can decide to abstain from voting or to vote for some choice $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C} \subseteq \mathbb{N}^{n_{\mathsf{components}}}$. In the latter case, the voter inputs c_i^{choice} to her voter-supporting device VSD_i, which proceeds as follows. First, VSD_i encrypts each (choice component) of c_i^{choice} separately under the public key pk and obtains a ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}) = (E_{\mathsf{pk}}(c_{1,i}^{\mathsf{component}}), \ldots, E_{\mathsf{pk}}(c_{n_{\mathsf{components}},i}^{\mathsf{component}}))$. That is, the j-th ciphertext in $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ encrypts the number of votes/points assigned by voter v_i to (choice) component j. After that, in addition to $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$, VSD_i creates a non-interactive zero-knowledge proof (NIZKP, see Appendix A.4) $\pi_i^{\mathfrak{C}}$ in order to prove that it knows which choice c_i^{choice} the vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ encrypts and that $c_i^{\mathsf{choice}} \in \mathfrak{C}$. Finally, VSD_i sends the ballot $\mathsf{b}_i = (id, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}), \pi_i^{\mathfrak{C}})$ to AS where id is an identifier of the election. The authentication server AS checks the ballot format and sends an acknowledgment to VSD_i.

2.1.2. Tallying Phase

After the voting phase is closed, the AS sends the list of ballots $\vec{\mathsf{b}}$ to the bulletin board BB. This list of ballots $\vec{\mathsf{b}}$ on BB is the input to the tallying phase. In the tallying phase, the trustees aggregate the votes. Each trustee T_k reads $\vec{\mathsf{b}}$ from the bulletin board BB and verifies its correctness (i.e., the trustee verifies that the tally does not include duplicates and invalid ballots). T_k homomorphically aggregates all vectors $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ in $\vec{\mathsf{b}}$ entry-wise and obtains a ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}}) = (E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_1, \ldots, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_{n_{\mathsf{components}}})$ with $n_{\mathsf{components}}$ entries each of which encrypts the number of votes/points of the respective (choice) component: $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_j = \bigoplus_{i=1}^{n_{\mathsf{voters}}} E_{\mathsf{pk}}(c_{j,i}^{\mathsf{component}})$ for $j \in \{1, \ldots, n_{\mathsf{components}}\}$. We call this the aggregated tally and denote the function aggregating individual votes with f^{agg}. The ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ now contains the number of votes each (choice) component received. The trustees verifiable decrypt $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ and publish the values $(c_{1,\mathsf{agg}}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},\mathsf{agg}}^{\mathsf{component}})$, together with a NIZKP π^{dec} proving the correctness of the decryption, on BB. With the public information on the number of votes for each (choice) component, everyone can calculate the election result function f^{res}, for example, by determining which candidate received the most votes.

2.1.3. Verifiability of Helios

Intuitively, Helios achieves verifiability in the following way. It employs NIZKPs to ensure ballot validity. The aggregation process does not involve confidential data; thus, anyone can double-check the results. The decryption of the aggregated tally is also made verifiable thanks to NIZKPs. Finally, the election result function is computed based on the decrypted aggregated tally; thus, anyone can recompute and confirm the result.

To protect the privacy of the voter's ballots Helios uses aggregation and does not decrypt individual ballots. However, as we show later in this thesis, this leads to a relatively low level of privacy.

In order to assess the security properties of a voting process, we need to establish a clear definition of voting processes within a computational framework. This will enable us to specify our security controls accurately. The following section will introduce a general computational model for which we will outline our security notions in the next section.

2.2. General Computational Model

To define security notions in the context of e-voting systems, we first need to define a framework to capture voting schemes. In order to do so, we model e-voting systems using the general computational model proposed in [CGK+16]. This model introduces processes, protocols, instances, and properties. In this section, we recall these terms.

2.2.1. Process

A *process* is a set of probabilistic polynomial-time interactive turing machines (ITMs, also called *programs*). Each ITM has named tapes (also called *channels*), which connect these ITMs in the process. A channel connects two programs if it appears in both ITMs but with opposite directions (input/output). A process may have external input/output channels, those that are not connected internally. At any time of a process run, only one program is active. The active program may send a message to another program via a channel. This program then becomes active and, after some computation, can send a message to another program, and so on. Each process contains a *master program*, which is the first program to be activated and also activated if the active program did not produce output (hence, did not activate another program). If the master program is active but does not produce output, a run stops.

We write a process $\mathfrak{P}$ as $\mathfrak{P} = \mathfrak{p}_1 \| \cdots \| \mathfrak{p}_n$, where $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ are programs. If $\mathfrak{P}_1$ and $\mathfrak{P}_2$ are processes (or programs), then $\mathfrak{P}_1 \| \mathfrak{P}_2$ is a process, provided that the processes are connectable: two processes are *connectable* if common external channels, i.e., channels with the same name, have opposite directions (input/output); we rename internal channels, if necessary. A process $\mathfrak{P}$ where all programs have the security parameter 1^η as input is denoted by $\mathfrak{P}^{(\eta)}$. In the processes we consider, the length of a run is always polynomially bounded in η. The random coins the programs use in $\mathfrak{P}$ uniquely determine the run.

2.2.2. Protocol

A *protocol* P is defined by a set of agents Σ (also called *parties* or *protocol participants*), and for each agent $\mathsf{a} \in \Sigma$, a program $\mathfrak{p}_\mathsf{a}$ which is supposed to be run by this agent. This program is the *honest program* of a. Agents are pairwise connected by channels; every agent has a channel to the adversary (see below).

Typically, a protocol contains a *scheduler* S as one of its participants, which acts as the master program of the protocol process (see below). The scheduler's task is to trigger the protocol participants and the adversary in the appropriate order. For example, in the context of e-voting, the scheduler would trigger protocol participants according to the phases of an election.

If $\mathfrak{p}_{\mathsf{a}_1}, \ldots, \mathfrak{p}_{\mathsf{a}_n}$ are the honest programs of the agents in P, then we denote the process $\mathfrak{p}_{\mathsf{a}_1} \| \ldots \| \mathfrak{p}_{\mathsf{a}_n}$ by $\mathfrak{P}_\mathsf{P}$.

The process $\mathfrak{P}_\mathsf{P}$ is always run with an adversary A. The adversary may run an arbitrary probabilistic polynomial-time program with channels to all protocol participants in $\mathfrak{P}_\mathsf{P}$. Hence, a *run* $\mathfrak{r}$ *of* P with adversary (adversary program) $\mathfrak{p}_\mathsf{A}$ is a run of the process $\mathfrak{P}_\mathsf{P} \| \mathfrak{p}_\mathsf{A}$. We consider $\mathfrak{P}_\mathsf{P} \| \mathfrak{p}_\mathsf{A}$ to be part of the description of $\mathfrak{r}$, such that it is always clear to which process, including the adversary, the run $\mathfrak{r}$ belongs.

The typical specification of honest programs of the agents of P is such that the adversary A can corrupt the programs by sending the special message `corrupt`. Upon receiving a `corrupt` message, the agent reveals all or some of its internal state to the adversary, and then the adversary takes over control of the agent. However, specific agents like schedulers are usually

immune to corruption and would ignore `corrupt` messages. Additionally, some agents may only accept `corrupt` messages during initialization, representing static corruption. This thesis' security analysis uses static corruption.

We specify that an agent $\mathtt{a}$ is *honest in a protocol run* $\mathfrak{r}$ if the agent is not corrupted in this run, i.e., has not accepted a `corrupt` message throughout the run. An agent $\mathtt{a}$ is *honest* if for all adversarial programs $\mathfrak{p}_{\mathsf{A}}$ the agent is honest in all runs of $\mathfrak{P}_{\mathsf{P}} \| \mathfrak{p}_{\mathsf{A}}$, i.e., $\mathtt{a}$ always ignores all `corrupt` messages.

2.2.3. Property

A *property* γ *of* P is a subset of the set of all runs of P. We denote the complement of γ by $\neg\gamma$. Recall that the description of a run $\mathfrak{r}$ of P contains the description of the process $\mathfrak{P}_{\mathsf{P}} \| \mathfrak{p}_{\mathsf{A}}$ (and hence, in particular the adversary) from which $\mathfrak{r}$ originates. Therefore, γ can be formulated independently of a specific adversary.

2.2.4. Negligible, overwhelming, δ-bounded

This thesis will need the notions of negligible, overwhelming, and δ-bounded functions, which we define in the following.

- A function f from the natural numbers to the interval $[0, 1]$ is *negligible* if, for every $c > 0$, there exists η_0 such that $f(\eta) \leq \frac{1}{\eta^c}$ for all $\eta > \eta_0$.
- The function f is *overwhelming* if the function $1 - f$ is negligible.
- A function f is δ-*bounded* if, for every $c > 0$ there exists η_0 such that $f(\eta) \leq \delta + \frac{1}{\eta^c}$ for all $\eta > \eta_0$.

2.3. Verifiability

Ensuring verifiability is essential for traditional paper-based elections and e-voting systems. Since e-voting systems are complex software and hardware systems, it is inevitable to encounter implementation errors, which can compromise the accuracy of the voting process. In addition, detecting intentional manipulation of these systems is often daunting, if not impossible. This poses a significant risk as the final election result may not accurately reflect the choices made by voters (see, e.g., the examples presented in Chapter 1, [WWH+10, SFD+14, Eps15, HLPT20] and references therein).

It is crucial to mitigate these risks to ensure the integrity of the democratic process. Therefore, a fundamental security property of elections is *verifiability* [Ben87, JCJ05, CCM08, Adi08, CCC+08, BBB+13, CGGI14, KZZ15a, KZZ15b, KMST16], i.e., voters should be able to verify that their votes were counted and every observer, including voters, election officials, and external observers, should be able to verify whether the final election outcome indeed corresponds to the votes submitted by the voters, even if voting devices and servers have implementation errors or are outright malicious.

Verifiability is deemed crucial by the Council of Europe in their recommendation on e-voting standards [Cou13]. The *Swiss Federal Chancellery Ordinance on Electronic Voting* [The13], the Estonian *Riigikogu Election Act* [Rii02], elections in Norway [Gjø11], and non-political elections in Germany [BSI21] also mandate individual verifiability requirements.

Numerous jurisdictions worldwide have aimed to implement online voting systems that offer verifiability [HAM22]. For instance, the Netherlands used an online voting system with individual verifiability in the 2004 *waterschappen* elections in Rijnland and Dommel [Pie10]. Norway employed an online voting system that was individually and universally verifiable in the 2011 and 2013 local elections [PCGK17]. In Estonia, individual verifiability has been supported since 2013, and universal verifiability has been implemented since 2017 [HMVW16].

In order to formally define verifiability of e-voting systems, we first recall the generic verifiability framework by Küsters, Truderung, and Vogt (*KTV framework*, originally presented in [KTV10] with some refinements in [CGK$^+$16]) that we use to analyze the verifiability level of the voting protocols presented in this thesis. Beyond its expressiveness, the KTV framework is particularly suitable to analyze our voting protocols because it does not make any specific assumptions on the choice space $\mathfrak{C}$ or the result function f^{res} of the voting protocol. Thus, we can apply the KTV verifiability definition without any further modifications.

2.3.1. Judge

The KTV verifiability definition assumes a *judge* J whose role is to accept or reject a protocol run by writing `accept` or `reject` on a dedicated channel $\mathsf{decision}_{\mathsf{J}}$. To make a decision, the judge runs a so-called *judging procedure*, which performs certain checks (depending on the protocol specification), such as verification of all zero-knowledge proofs (see Appendix A.4). Intuitively, J accepts a run if the protocol run looks as expected. The input to the judge is solely public information, including all information and complaints (e.g., by voters) posted on the bulletin board. Therefore, the judge can be considered a *virtual* entity: the judging procedure can be carried out by any party, including external observers and voters.

2.3.2. Goal

The KTV verifiability definition of a protocol centers around the notion of a *goal*. Formally, a goal is simply a property γ of the protocol, i.e., a set of runs (see Section 2.2). Intuitively, such a goal specifies *correct* runs in some protocol-specific sense. For e-voting, the goal would contain those runs where the announced election result corresponds to the voters' actual choices. In what follows, we present the goal $\gamma(k, \varphi)$ proposed in [CGK$^+$16] that we apply in this thesis. The parameter k is a non-negative integer, and φ is a formula that precisely models the trust assumptions. Roughly speaking, the goal $\gamma(k, \varphi)$ contains all those runs in which the distance between the produced result and the *ideal* one (obtained when counting the actual choices of honest voters and one valid choice for every dishonest voter) is bounded by k under the assumption that φ holds.

More precisely, we consider a specific distance function f_{dist}. In order to define f_{dist}, we first define a function $f_{\mathsf{count}} \colon \mathfrak{C}^n \to \mathbb{N}^{|\mathfrak{C}|}$ which, for a vector $(c_1^{\mathsf{choice}}, \ldots, c_n^{\mathsf{choice}}) \in \mathfrak{C}^n$ (representing a multiset of voters' choices), counts how many times each choice occurs in this vector. For example, $f_{\mathsf{count}}(A, A, C)$ assigns 2 to A, 1 to C, and 0 to all the remaining choices. Now, for two vectors of choices $\overrightarrow{c_0^{\mathsf{choice}}}, \overrightarrow{c_1^{\mathsf{choice}}}$, the distance function f_{dist} is defined by

$$f_{\mathsf{dist}}(\overrightarrow{c_0^{\mathsf{choice}}}, \overrightarrow{c_1^{\mathsf{choice}}}) = \sum_{c_i^{\mathsf{choice}} \in \mathfrak{C}} \left| f_{\mathsf{count}}(\overrightarrow{c_0^{\mathsf{choice}}})[c_i^{\mathsf{choice}}] - f_{\mathsf{count}}(\overrightarrow{c_1^{\mathsf{choice}}})[c_i^{\mathsf{choice}}] \right|.$$

For example, $f_{\mathsf{dist}}((A, A, C), (A, B, C, C)) = 3$.

Now, let $f^{\mathrm{res}} \circ f^{\mathsf{agg}} \colon \mathfrak{C}^* \to \{0,1\}^*$ be the result function of the election and consider a protocol run $\mathfrak{r}$. With $\mathsf{V}_{\mathsf{honest}}$, we denote the set of honest voters in $\mathfrak{r}$, i.e., the set of voters that did not receive a `corrupt` message from the adversary in the run $\mathfrak{r}$. Likewise, we define with $\mathsf{V}_{\mathsf{dishonest}}$ the set of dishonest voters, i.e., the set of voters that did receive a `corrupt` message from the adversary in run $\mathfrak{r}$. As explained in Section 2.2, in this thesis, we use static corruption.

Then, the goal $\gamma(k, \varphi)$ is satisfied in $\mathfrak{r}$ if either (a) the trust assumption φ does not hold true in $\mathfrak{r}$, or if (b) φ holds true in $\mathfrak{r}$ and there exist valid choices $(c_i^{\mathsf{choice}})_{i \in \mathsf{V}_{\mathsf{dishonest}}}$ (representing possible choices of dishonest voters) and choices $\overrightarrow{c_{\mathsf{real}}^{\mathsf{choice}}} = (c_i^{\mathsf{choice,real}})_{i \leq n_{\mathsf{voters}}}$ such that:

(i) an election result is published, and it is equal to $f^{\mathrm{res}} \circ f^{\mathsf{agg}}(\overrightarrow{c_{\mathsf{real}}^{\mathsf{choice}}})$, and

(ii) $f_{\mathsf{dist}}(\overrightarrow{c_{\mathsf{ideal}}^{\mathsf{choice}}}, \overrightarrow{c_{\mathsf{real}}^{\mathsf{choice}}}) \leq k$,

where $\overrightarrow{c_{\mathsf{ideal}}^{\mathsf{choice}}}$ consists of the actual choices $(c_i^{\mathsf{choice}})_{i \in \mathsf{T}_{\mathsf{honest}}}$ made by the honest voters and the possible choices $(c_i^{\mathsf{choice}})_{i \in \mathsf{V}_{\mathsf{dishonest}}}$ made by the dishonest voters.

Note that when an adversary *drops* one honest vote, this increases the distance in condition (ii) by one, but when he *replaces* an honest vote by another one, this increases the distance by two. The replacement corresponds to the actual effect of manipulation on the final result.

2.3.3. Definition of Verifiability

The concept of the verifiability definition is as follows. The judge J should accept a run only if it meets the goal $\gamma(k, \varphi)$; hence, the published election result corresponds to the voters' actual choices. More precisely, the definition requires that the probability (over the set of all protocol runs) that the goal $\gamma(k, \varphi)$ is not satisfied, but the judge accepts the run is δ-bounded. Although $\delta = 0$ is desirable, this would be too strong for almost all e-voting protocols. For example, not all voters typically check whether their ballot appears on the bulletin board, giving an adversary A the opportunity to manipulate or drop some ballots without being detected. Therefore, we generally cannot achieve $\delta = 0$.

By $\Pr[\mathfrak{P}^{(\eta)} \mapsto (\mathsf{J}\colon \texttt{accept})]$ we denote the probability that $\mathfrak{P}$, with security parameter 1^η, produces a run which is accepted by J. Analogously, by $\Pr[\mathfrak{P}^{(\eta)} \mapsto \neg\gamma, (\mathsf{J}\colon \texttt{accept})]$ we denote

the probability that $\mathfrak{P}$, with security parameter 1^η, produces a run which is not in γ but nevertheless accepted by J.

Definition 2.1 (Verifiability)**.** *Let* P *be a protocol with a set of agents* Σ*. Let* $\delta \in [0,1]$ *be the tolerance,* $\mathsf{J} \in \Sigma$ *be the judge, and* γ *be a goal. Then, we define that* the protocol P is (γ,δ)-verifiable by the judge J *if for all adversaries* $\mathfrak{P}_\mathsf{A}$ *and* $\mathfrak{P} = (\mathfrak{p}_\mathsf{P} \| \mathfrak{P}_\mathsf{A})$*, the probability*

$$\Pr[\mathfrak{P}^{(\eta)} \mapsto \neg\gamma, (\mathsf{J}\colon \texttt{accept})]$$

is δ*-bounded as a function of* η*.*

A protocol P could trivially satisfy verifiability with a judge who never accepts a run. Therefore, one would also require a *soundness* or *fairness* condition. At the very least, one would expect that if the protocol runs with an adversary, which would not corrupt parties, then the judge accepts a run. Formally, for such an adversary $\mathfrak{P}_\mathsf{A}$, we require that $\Pr[\mathfrak{P}^{(\eta)} \mapsto (\mathsf{J}\colon \texttt{accept})]$ is overwhelming.

2.4. Accountability

A closely related and stronger property than verifiability is *accountability* [KTV10], which some systems also provide. Accountability requires that misbehavior leading to incorrect election results be detected and that misbehaving parties can be held accountable.

This section provides an overview of the general definition of accountability [KTV10], independent of any domain, and its specific application to e-voting. According to the paper, verifiability is a weaker form of accountability. In the case of verifiability, if the protocol fails to achieve a specific goal, the judge will not accept such a run, but misbehaving parties are not necessarily blamed. On the other hand, accountability requires the judge to hold misbehaving parties responsible.

As emphasized in the introduction and other work (e.g., [KTV10]), e-voting schemes must ensure accountability, not just verifiability. Accountability demands that the judge blames those participants who misbehave, or at least some of them, in case a run fails to achieve the desired goal of the protocol due to the misbehavior of one or more protocol participants.

In the case of voting protocols, a desired goal is to ensure that the published election result corresponds to the votes the voters cast. Accountability guarantees that if the published result of the election does not match the votes cast by the voters (due to the misbehavior of one or more protocol participants), then the judge holds one or more participants accountable.

We use the following notions of verdicts and accountability properties to specify accountability.

2.4.1. Verdicts

The judge can give a verdict (on a dedicated output channel) and state which parties to blame (that is, which ones, according to the judge, have misbehaved). In the simplest case, a verdict

can state that a specific party misbehaved (behaved dishonestly). Such an atomic verdict is denoted by $\mathsf{dis}(\mathsf{a})$. It is also useful to state more fine-grained or weaker verdicts, such as a_i *or* a_j *misbehaved*. Therefore, in the general case, we will consider verdicts, which are boolean combinations of atomic verdicts. In fact, in our formal analysis carried out in this thesis, we use in some cases verdicts of the form $\mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{AS})$ stating that either the i-th voter v_i or the authentication server AS misbehaved (but the verdict leaves open, as it might not be clear, which one of them). Given the process $\mathfrak{P}_\mathsf{P}$ of a protocol P, a verdict ψ *is true in* $\mathfrak{P}_\mathsf{P}$, written $\mathfrak{P}_\mathsf{P} \models \psi$, if and only if the formula ψ evaluates to true with the proposition $\mathsf{dis}(\mathsf{a})$ set to false if a is honest in $\mathfrak{P}_\mathsf{P}$ (as defined in Section 2.2), and set to true otherwise.

2.4.2. Accountability Constraints

An *accountability constraint* is a tuple $(\alpha, \psi_1, \ldots, \psi_n)$, written $(\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_n)$, where α is a property of P (recall that, formally, this is a set of runs of $\mathfrak{P}_\mathsf{P}$) and $\psi_1, \ldots, \psi_n$ are verdicts. Such a constraint *covers* a run $\mathfrak{r}$ if $\mathfrak{r} \in \alpha$. Intuitively, in a constraint $\Gamma = (\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_n)$ the set α contains runs in which some desired goal of the protocol is *not* met (due to the misbehavior of some protocol participants). The formulas $\psi_1, \ldots, \psi_n$ are the possible (minimal) verdicts that are supposed to be stated by J in such a case; J is free to state stronger verdicts. Formally, for a run $\mathfrak{r}$, J *ensures* Γ *in* $\mathfrak{r}$, if either $\mathfrak{r} \notin \alpha$ or J states a verdict ψ in $\mathfrak{r}$ that implies one of the verdicts $\psi_1, \ldots, \psi_n$ (in the sense of propositional logic).

2.4.3. Individual Accountability

In practice, so-called *individual accountability* is highly desirable to deter parties from misbehaving. Formally, $(\alpha \Rightarrow \psi_1 \mid \cdots \mid \psi_n)$ provides *individual accountability*, if for every $i \in \{1, \ldots, n\}$, there exists a party a such that ψ_i implies $\mathsf{dis}(\mathsf{a})$. In other words, $\psi_1, \ldots, \psi_n$ determines at least one misbehaving party.

2.4.4. Accountability Property

A set Φ of accountability constraints for a protocol P is called an *accountability property* of P. One should define an accountability property Φ in such a way that it covers all relevant cases in which a desired goal for P is not met, i.e., whenever some desired goal of P is not satisfied in a given run $\mathfrak{r}$ due to some misbehavior of some protocol participants, then there exists a constraint in Φ which covers $\mathfrak{r}$. In particular, in this case, the judge must state a verdict.

2.4.5. Definition of Accountability

Let P be a protocol with the set of agents Σ and an accountability property Φ of P. Let $\mathfrak{P}_\mathsf{P}$ be the process of P and $\mathsf{J} \in \Sigma$ be an agent of P. We write $\Pr[\mathfrak{P}_\mathsf{P}^{(\eta)} \mapsto \neg(\mathsf{J}\colon \Phi)]$ to denote the probability that $\mathfrak{P}_\mathsf{P}$, with security parameter 1^η, produces a run such that J does not ensure Γ in this run for some $\Gamma \in \Phi$.

Definition 2.2 (Accountability)**.** *Let* P *be a protocol with a set of agents* Σ*. Let* $\delta \in [0,1]$ *be the tolerance,* $\mathsf{J} \in \Sigma$ *be the judge, and* Φ *be an accountability property of* P*. Then, we define that* the protocol P is (Φ, δ)-accountable w.r.t. the judge J *if for all adversaries* $\mathfrak{P}_\mathsf{A}$ *and* $\mathfrak{P}_\mathsf{P} = (\mathfrak{P}_{\mathsf{a}_1} \| \dots \| \mathfrak{P}_{\mathsf{a}_n} \| \mathfrak{P}_\mathsf{A})$*, the probability*

$$\Pr[\mathfrak{P}_\mathsf{P}^{(\eta)} \mapsto \neg(\mathsf{J}\colon \Phi)]$$

is δ*-bounded as a function of* η*.*

Similarly to the verifiability definition, we also require that the judge J is *computationally fair* in P, i.e., for $\mathfrak{P}_\mathsf{P}$ corresponding to P, the judge J states false verdicts only with negligible probability.

2.5. Privacy

Ensuring the privacy of votes is crucial for democratic elections. It is essential to protect individuals' autonomy to vote based on their beliefs and values without fear of coercion or retribution. The secrecy of the ballot helps maintain the integrity of the electoral process, enabling citizens to vote freely and independently and preventing vote-buying and coercion. When voters know their choices remain confidential, they are more likely to vote and trust the electoral process, encouraging participation.

In this section, we define the privacy of e-voting schemes. Our privacy result states that no adversary can distinguish whether a voter (called the *voter under observation*) voted for c_0^{choice} or c_1^{choice} when running the honest voter program.

In what follows, we introduce the class of adversaries we consider and then present our privacy definition.

2.5.1. Risk-Avoiding Adversaries

An adversary with control over the authentication server could manipulate the election by discarding or altering all ballots except for the voter's ballot under observation, compromising the privacy of the voters' choice.

However, such an attack involves a risk: Recall that the probability of being caught grows exponentially in k of manipulated honest votes (see Section 2.3). Thus, in the above attack where k is significant, the probability of the adversary getting caught would be close to 1. In the context of e-voting, caught misbehaving parties have to face severe penalties or loss of reputation; therefore, this attack seems completely unreasonable.

A more reasonable adversary could consider dropping some small number of votes, for which the degree of being caught is not that huge, to weaken privacy to some degree. We use the

notion of *k-risk-avoiding adversaries* (initially introduced in [KMST16]) to analyze this trade-off. Intuitively, a k-risk-avoiding adversary produces runs in which the goal $\gamma(k, \varphi)$ holds (see Section 2.3) for those voters who did not abstain from voting. In other words, the distance between the *ideal* input (provided by the honest voters who submitted a ballot, plus one choice per dishonest voter) and the *real* one is bounded by k.

More precisely, we define a goal $\gamma'(k, \varphi)$ exactly in the same way as $\gamma(k, \varphi)$, except that when measuring the distance, we only take into account valid choices that are not $\mathsf{abstain}$, i.e.,

$$f_{\mathsf{dist}}(\overrightarrow{c_0^{\mathsf{choice}}}, \overrightarrow{c_1^{\mathsf{choice}}}) = \sum_{c_i^{\mathsf{choice}} \in \mathfrak{C}'} \left| f_{\mathsf{count}}(\overrightarrow{c_0^{\mathsf{choice}}})[c_i^{\mathsf{choice}}] - f_{\mathsf{count}}(\overrightarrow{c_1^{\mathsf{choice}}})[c_i^{\mathsf{choice}}] \right|,$$

where $\mathfrak{C}' = \mathfrak{C} \setminus \{\mathsf{abstain}\}$. An adversary is *k-risk-avoiding in a run of* P if the property $\gamma'(k, \varphi)$ is satisfied. An adversary (of the process $\mathfrak{P}_{\mathsf{P}}$ of P) is *k-risk-avoiding* if it is k-risk-avoiding with overwhelming probability (over the set of all runs of $\mathfrak{P}_{\mathsf{P}}$).

Since $\gamma(k, \varphi)$ is stronger than $\gamma'(k, \varphi)$, the accountability property guarantees that each adversary who is not k-risk-avoiding gets caught with probability $1 - \delta_k(p_{\mathsf{verify}})$ which converges exponentially fast to 1. Here, with $p_{\mathsf{verify}} \in [0, 1]$, we denote the probability that an honest voter v_i performs the necessary checks to verify that her ballot was included correctly in the tally. Therefore, assuming that the adversary knows this risk, it is reasonable for him to be k-risk-avoiding if he does not want to be held accountable. However, increasing k does not help the adversary much in breaking privacy, as argued below.

2.5.2. Definition of Privacy

In the analysis carried out in this thesis, we use the privacy definition for e-voting protocols proposed in [KTV11], which allows us to *measure* the level of privacy a protocol provides (as opposed to binary privacy notions, such as [KMW12]).

As briefly mentioned above, we formalize the privacy of an e-voting protocol as the inability of an adversary to distinguish whether some voter $\mathsf{v}_{\mathsf{obs}}$ (the voter under observation), who runs her honest program $\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}$, voted for c_0^{choice} or c_1^{choice}.

To define this notion formally, we first introduce the following notation. Let P be an e-voting protocol (in the sense of Section 2.2 with voters, authorities, choice space, result function, and the other voting components). Given a voter $\mathsf{v}_{\mathsf{obs}}$ and $c_i^{\mathsf{choice}} \in \mathfrak{C}$, the protocol P induces a process of the form $(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_i^{\mathsf{choice}}) \| \mathfrak{P}^*)$ where $\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_i^{\mathsf{choice}})$ is the honest program of the voter $\mathsf{v}_{\mathsf{obs}}$ under observation which takes c_i^{choice} as the choice for which $\mathsf{v}_{\mathsf{obs}}$ votes and where $\mathfrak{P}^*$ is the composition of programs of the remaining parties. In the case of the voting schemes in this thesis, $\mathfrak{P}^*$ would include the scheduler, the bulletin board, the authentication server, all other voters, and all trustees.

Let $\Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_i^{\mathsf{choice}}) \| \mathfrak{P}^*)^{(\eta)} \mapsto 1]$ denote the probability that the adversary, i.e., the dishonest agents in $\mathfrak{P}^*$, writes the output 1 on some dedicated channel in a run of $(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_i^{\mathsf{choice}}) \| \mathfrak{P}^*)$ with

security parameter η and some $c \in^{\mathsf{choice}} \mathfrak{C}$, where we take the probability over the random coins used by the agents in $(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_i^{\mathsf{choice}}) \| \mathfrak{P}^*)$.

Now, we define privacy for k-risk-avoiding adversaries.

Definition 2.3 (Privacy)**.** *Let* P *be a protocol with a voter under observation* $\mathsf{v}_{\mathsf{obs}}$ *and let* $\delta \in [0,1]$*. We define that* P with $n_{\mathsf{voters}}^{\mathsf{honest}}$ honest voters achieves δ-privacy w.r.t. k-risk-avoiding adversaries*, if for all* $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \in \mathfrak{C} \setminus \{\mathsf{abstain}\}$*, all programs* $\mathfrak{P}^*$ *of the remaining parties such that at least* $n_{\mathsf{voters}}^{\mathsf{honest}}$ *are honest in* $\mathfrak{P}^*$ *and such that the adversary is* k*-risk-avoiding, we have that*

$$|\Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_0^{\mathsf{choice}}) \| \mathfrak{P}^*)^{(\eta)} \mapsto 1] - \Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_1^{\mathsf{choice}}) \| \mathfrak{P}^*)^{(\eta)} \mapsto 1]|$$

is δ*-bounded as a function of* η*.*

The requirement $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \neq \mathsf{abstain}$ allows the adversary to distinguish whether a voter voted.

Since δ typically depends on the number $n_{\mathsf{voters}}^{\mathsf{honest}}$ of honest voters, we formulate privacy w.r.t. this number. Note that a smaller δ means a higher level of privacy. However, even for an ideal e-voting protocol, where voters enter their votes in secret, and the adversary sees only the election outcome, δ cannot be 0: there is, for example, a non-negligible chance that all honest voters, including the voter under observation, voted for the same candidate, in which case the adversary can see how the voter under observation voted. We denote the privacy level of the ideal protocol with the result function f^{res} by $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$, where $n_{\mathsf{voters}}^{\mathsf{honest}}$ is the number of honest voters and μ the probability distribution used by honest voters to determine their choices.

2.6. Choice Spaces

Numerous voting methods exist, each offering its unique approach to capturing and representing the will of the voters. From simple single choice to more complex voting methods like ranking candidates, these methods provide different mechanisms to reflect voters' preferences and allow various election result functions.

We formalize a voting method using a *choice space* $\mathfrak{C}$, a set of all possible choices from which the voters can choose. The choice space $\mathfrak{C}$ determines the valid choices for voters to cast ballots. As explained in Section 2.1, the parameter $n_{\mathsf{components}} \in \mathbb{N}$ denotes the number of *(choice) components* of an election and possible voting choices $\mathfrak{C} \subseteq \mathbb{N}^{n_{\mathsf{components}}}$. Both parameters, $n_{\mathsf{components}}$ and $\mathfrak{C}$, are instantiated depending on the voting method.

In this section, we present various choice spaces supported by the election systems we present in this thesis.

2.6.1. Single Choice

Single choice, also known as *plurality voting* or *first-past-the-post*, is a voting method where each voter is allowed to select exactly one option from the available choices. Many elections, particularly in systems prioritizing simplicity and ease of understanding, use single-choice voting. However, single-choice voting can sometimes lead to outcomes that do not accurately represent the overall preferences of the electorate, especially in situations with multiple candidates or diverse viewpoints. The choice space $\mathfrak{C}_{\mathsf{Single}}$ for single-vote is given by

$$\mathfrak{C}_{\mathsf{Single}} = \left\{ (x_1, \ldots, x_{n_{\mathsf{cand}}}) \mid x_i \in \{0,1\}, \sum_{i=1}^{n_{\mathsf{cand}}} x_i = 1 \right\}$$

For this choice space, we have $n_{\mathsf{components}} = n_{\mathsf{cand}}$.

2.6.2. Multiple Choice

Multiple-choice voting (also called *multi-vote*) is a voting method that allows voters to select more than one option from the available choices. Using this voting method, each voter can indicate their preferences by choosing multiple candidates or options, typically within a predefined limit. Multiple-choice voting allows voters to support multiple candidates or express nuanced preferences. It can be advantageous for filling multiple positions, such as electing multiple representatives or ranking preferences. This method allows for a more diverse range of voices and can help ensure that the elected outcome better reflects the varied perspectives and preferences of the voters than a single choice. However, it can lead to vote-splitting or strategic voting, where voters concentrate their support on a limited number of candidates to maximize the chances of their preferred choices winning. In the general case of multi-vote, a voter has to distribute a previously defined number n_{votes} of votes among the candidates, with the restriction that only a bounded number b of votes is allowed to be given to each candidate. The following choice space captures multiple-choice voting:

$$\mathfrak{C}_{\mathsf{Multi},\mathfrak{o}_{\mathsf{comp}},n_{\mathsf{votes}},b} := \Bigg\{ (x_1, \ldots, x_{n_{\mathsf{components}}}) \mid x_i \in \{0,1,\ldots,b\}, \mathfrak{o}_{\mathsf{comp}} \in \{=,\leq\}, \sum_{i=1}^{n_{\mathsf{components}}} x_i \; \mathfrak{o}_{\mathsf{comp}} \; n_{\mathsf{votes}} \Bigg\}.$$

If $\mathfrak{o}_{\mathsf{comp}}$ is to equal ($=$), the voters most distribute n_{votes} votes. If we set $\mathfrak{o}_{\mathsf{comp}}$ to less than or equal to ($\leq$) instead, the voters can distribute up to n_{votes}. The most common use is $b = 1$, i.e., voters can only give one vote to each candidate.

2.6.3. Borda

Borda voting is a prominent ranked voting method. The Eurovision Song Contest (ESC) uses Borda voting to determine the winner of its grand final [Eur20]. Also, national elections, e.g., parliamentary elections in the Republic of Nauru [Rep16] apply Borda voting. We present both elections in Section 2.8. In Borda voting, a voter ranks the candidates according to her preferences, and each candidate receives several points that depend on the position in the ranking. There are many options to determine these points. Here, we present the two most famous options: the *point list* variant and the *tournament style* variant. While ties are typically only allowed for the last place in the ranking of Borda elections, we generalize these such that arbitrary ties are allowed in both variants. All Borda variants aggregate the points for each candidate, and any result function can be applied to determine the election winner (for example, the candidate with the most points wins).

2.6.3.1. Point List

The *point list* ballot format works as follows. Each rank corresponds to a unique number of points defined via a list $\mathcal{P} \in \mathbb{N}^{n_{\mathsf{points}}}$, wherein the standard case, $n_{\mathsf{points}} = n_{\mathsf{cand}} = n_{\mathsf{components}}$ is the number of candidates and $\mathcal{P}[1]$ is the number of points assigned to the candidate that is ranked first and so on. The standard version of Borda requires candidates to be assigned a unique rank (i.e., a unique number of points); advanced variants allow for assigning multiple candidates the same rank, i.e., the same number of points. In that case, if i candidates are assigned the same rank r, then the next lower candidate is required to start at rank $r+i$ (instead of $r+1$, as is the case when ranks are assigned uniquely).

The choice space $\mathfrak{C}_{\mathsf{BordaPointList}}(\mathcal{P})$ captures the above behavior, where we interpret $\mathcal{P}$ as a list and a set. Note that for simplicity, we assume that $n_{\mathsf{points}} = n_{\mathsf{cand}}$.

$$\mathfrak{C}_{\mathsf{BordaPointList}}(\mathcal{P}) = \Bigg\{ (x_1, \ldots, x_{n_{\mathsf{cand}}}) \;\Bigg|\; \forall i : x_i \in \mathcal{P} \;\wedge \forall r \in \{1, \ldots, n_{\mathsf{points}}\}, \forall i \in \{2, \ldots, \sigma(r)\} : \nexists j \in \{1, \ldots, n_{\mathsf{cand}}\} : x_j = \mathcal{P}[r+i-1] \wedge \Big(r + \sigma(r) \leq n_{\mathsf{points}} \Rightarrow \exists j \in \{1, \ldots, n_{\mathsf{cand}}\} : x_j = \mathcal{P}[r+\sigma(r)] \Big) \Bigg\},$$

where $\sigma(r) \in \{1, \ldots, n_{\mathsf{cand}}\}$ denotes the number of occurrences of $\mathcal{P}[r]$ in $(x_1, \ldots, x_{n_{\mathsf{cand}}})$. Using $\mathfrak{C}_{\mathsf{BordaPointList}}(\mathcal{P})$, standard Borda can be derived by additionally requiring $\sigma(r) = 1$ for all values of r. Further straightforward variations include allowing the voter to submit only a partial ranking.

2.6.3.2. Tournament Style

The *tournament style* ballot format works as follows. Based on the ranking of the candidates by a voter, each candidate receives two points per candidate that is ranked lower and one point for each other candidate with the same rank. The following choice space captures this:

$$\mathfrak{C}_{\mathsf{BordaTournamentStyle}} = \Big\{(x_1,\dots,x_{n_{\mathsf{cand}}}) \;\Big|\; \exists (r_1\dots,r_{n_{\mathsf{cand}}}) \in \mathbb{N}^{n_{\mathsf{cand}}} \text{ s.t. } \forall i \in \{1,\dots,n_{\mathsf{cand}}\} : \\ x_i = 2\cdot|\{j \in \{1,\dots,n_{\mathsf{cand}}\} : r_j < r_i\}| \\ + |\{j \in \{1,\dots,n_{\mathsf{cand}}\} \setminus \{i\} : r_j = r_i\}|\Big\}.$$

2.6.4. Grading

Majority judgment is an alternative voting method designed to capture more nuanced preferences of voters [BL07]. In this method, instead of simply selecting a single candidate or option, voters assess each candidate on a predefined scale of *grades*, typically ranging from *excellent* to *poor* or *acceptable* to *unacceptable*. Voters assign judgments to each candidate based on their evaluation.

Majority judgment promotes a more comprehensive understanding of voter preferences and allows for a more fine-grained evaluation of candidates. It allows voters to express nuanced opinions and preferences, considering various aspects of each candidate's qualities or performance. This method emphasizes the evaluation of candidates rather than a direct comparison or ranking of preferences. However, implementing majority judgment in large-scale elections can present challenges, such as accurately aggregating and interpreting the judgments.

In order to allow for aggregation of such ballots, the ballots are a matrix of the candidates and grades, where entry A_{ij} is set to one if candidates i receives grade j, and zero otherwise. The choice space $\mathfrak{C}_{\mathsf{MajorityJudgment}}$ captures this:

$$\mathfrak{C}_{\mathsf{MajorityJudgment}} = \left\{A \in \{0,1\}^{n_{\mathsf{cand}}\times n_{\mathsf{grades}}} \;\middle|\; \forall i \in \{1,\dots,n_{\mathsf{cand}}\} : \sum_{j\in\{1,\dots,n_{\mathsf{grades}}\}} A_{ij} = 1\right\}$$

2.6.5. Duel Matrix

Voters are allowed to compare the candidates to each other. That is, for every pair of candidates $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}}$, a so-called *duel* takes place, where the voter can decide whether she prefers c_i^{cand} or c_j^{cand}.

Such options are captured using a *duel matrix* $\mathsf{DM} \in \{0,1\}^{n_{\mathsf{cand}} \times n_{\mathsf{cand}}}$, which denotes in each entry DM_{ij} whether the voter prefers c_i^{cand} over c_j^{cand} ($\mathsf{DM}_{ij} = 1$ if that is the case, $\mathsf{DM}_{ij} = 0$ otherwise). We define the choice space $\mathfrak{C}_{\mathsf{DuelMatrix}}$ as follows:

$$\mathfrak{C}_{\mathsf{DuelMatrix}} = \Big\{ \mathsf{DM} \in \{0,1\}^{n_{\mathsf{cand}} \times n_{\mathsf{cand}}} \;\Big|\; \forall i,j \in \{1,\dots,n_{\mathsf{cand}}\} : i \neq j \implies \mathsf{DM}_{ij} + \mathsf{DM}_{ji} = 1 \Big\}.$$

These duel matrices of the voters $\mathsf{DM}^1, \dots, \mathsf{DM}^{n_{\mathsf{voters}}}$ can be aggregated entry-wise to obtain the aggregated duel matrix $\mathsf{DM}^{\mathsf{agg}}$, i.e., $\mathsf{DM}_{ij}^{\mathsf{agg}} := \sum_{k=1}^{n_{\mathsf{voters}}} \mathsf{DM}_{ij}^k$. Therefore, $\mathsf{DM}_{ij}^{\mathsf{agg}}$ denotes how often candidate i was preferred over candidate j.

This choice space sets $n_{\mathsf{components}}$ to $n_{\mathsf{cand}} \cdot n_{\mathsf{cand}}$. In particular, the number of (choice) components is quadratic in the number of candidates.

Several sophisticated result functions exist to select a suitable election result based on this choice space. We will explain them in Section 2.7.

2.6.6. Ranking Matrix

Allowing the voters to rank the candidates is considered a more comprehensive method than other voting systems like plurality voting, as it considers the relative preferences of voters for all candidates rather than just the top choice. While the duel matrix does not take transitivity into account, e.g., if a voter prefers c_i^{cand} over c_j^{cand}, and c_j^{cand} over c_k^{cand}, then this does not imply that the voter does prefer c_i^{cand} over c_k^{cand}. In order to create a ranking of the candidates, we allow transitivity to obtain a ranking matrix ψ. We define the choice space $\mathfrak{C}_{\mathsf{RankingMatrix}}$ as follows:

$$\begin{aligned}\mathfrak{C}_{\mathsf{RankingMatrix}} = \Big\{ \mathsf{DM} \in \{0,1\}^{n_{\mathsf{cand}} \times n_{\mathsf{cand}}} \;\Big|\; \forall i,j,k \in \{1,\dots,n_{\mathsf{cand}}\} : i \neq j \implies \\ \big(\mathsf{DM}_{ij} + \mathsf{DM}_{ji} = 1 \;\; \wedge \;\; \mathsf{DM}_{ij} = 1 \wedge \mathsf{DM}_{jk} = 1 \Rightarrow \mathsf{DM}_{ik} = 1\big) \Big\}.\end{aligned}$$

Just as the duel matrices, these ranking matrices of the voters $\psi^1, \dots, \psi^{n_{\mathsf{voters}}}$ can be aggregated entry-wise to obtain the aggregated ranking matrix Ψ, i.e., $\Psi_{ij} := \sum_{k=1}^{n_{\mathsf{voters}}} \psi_{ij}^k$. Therefore, Ψ_{ij} denotes how often candidate i was preferred over candidate j.

This choice space sets $n_{\mathsf{components}}$ to $n_{\mathsf{cand}} \cdot n_{\mathsf{cand}}$. In particular, the number of (choice) components is quadratic in the number of candidates.

Again, several sophisticated result functions exist to select a suitable election result based on this choice space. We will explain them in Section 2.7.

2.6.7. Ranking Permutation

In comparison to the ranking matrix, another way to capture a ranking of candidates is to use a permutation of the list of candidates.

n_{cand}	# Rankings	
	Complete	Partial
1	1	2
2	2	5
3	6	16
4	24	65
5	120	326
6	720	1,957
7	5,040	13,700
8	40,320	109,601
9	362,880	986,410
10	3,628,800	9,864,101

Table 2.1.: Number of complete and partial rankings of n_{cand} many candidates.

We interpret a ranking as a (potentially partial) permutation of the n_{cand} candidates and consider a vote as a vote for a particular permutation. Thus, we can represent a ballot to a ranking as a single-choice ballot that selects a permutation of the candidates.

If we only allow complete rankings (the voter has to rank all candidates), there are $n_{\mathsf{cand}}!$ many possible rankings. However, if we allow partial rankings (a ranking of a freely selected subset of the candidates), there are $\sum_{i=0}^{n_{\mathsf{cand}}} \frac{n_{\mathsf{cand}}!}{(n_{\mathsf{cand}}-i)!}$ possible rankings. We present the number of possible complete and partial rankings on various numbers of candidates in Table 2.1. We remark that using this approach, the number of (choice) components grows factorially in the number of candidates.

2.7. Election Result Functions

Election result functions determine the outcome of the election. Different election result functions yield distinct interpretations of voter preferences and influence the composition of elected bodies. Examples of election result functions include simple plurality, where the candidate with the most votes wins, and proportional representation, which allocates seats based on each party's votes. Other functions, such as runoff elections and Condorcet methods, introduce more complex mechanisms for ranking and weighting voter preferences. The diversity of election result functions allows for flexibility in designing electoral systems that align with specific goals and values.

Formally, an *election result function* is a deterministic polynomial election result function $f^{\text{res}} : \mathfrak{C}^* \to \{0,1\}^*$ that computes the election outcome based on the voters' votes. Usually, the function outputs a list of indices $(b_1, \ldots, b_{n_{\mathsf{cand}}}), b \in \{0,1\}$ that indicates for each candidate (or

party, depending on the context) c_i^{cand} whether this candidate won the election (which might have different meanings depending on the concrete election result function).

The complexity of evaluating election result functions based on individual votes scales in the number of voters, which is inefficient in typical settings with more voters than the number of candidates. Therefore, we usually make use of homomorphic aggregation. That is, we homomorphically aggregate all ballots c_j^{choice} in $(c_j^{\mathsf{choice}})_{j=1}^{n_{\mathsf{voters}}}$ entry-wise to obtain a vector $c_{\mathsf{agg}}^{\mathsf{choice}} = (c_{1,\mathsf{agg}}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},\mathsf{agg}}^{\mathsf{component}})$ with $n_{\mathsf{components}}$ entries each of which contains the number of votes/points of the respective (choice) component: $c_{i,\mathsf{agg}}^{\mathsf{component}} = \sum_{j=1}^{n_{\mathsf{voters}}} c_{i,j}^{\mathsf{component}}$ for $i \in \{1, \ldots, n_{\mathsf{components}}\}$. As in Section 2.1, we denote the function that aggregated the individual votes with f^{agg}. Then, we use this aggregated tally $c_{\mathsf{agg}}^{\mathsf{choice}}$ to compute the election result. With this, the complexity no longer depends on the number of voters since we can now define $f^{\mathrm{res}} : (c_{i,\mathsf{agg}}^{\mathsf{component}})_{i=1}^{n_{\mathsf{components}}} \to \{0,1\}^*$.

As presented in Section 2.1, voting schemes typically use aggregation to increase vote privacy: E-voting systems that follow the structure of Helios decrypt and publish this aggregated tally, such that every party can compute the election result function in plain based on the aggregated tally. If one would instead decrypt and publish the individual ballots, everyone could see how each voter voted. However, employing tally-hiding, we only reveal the election result. The following election result functions all start with the aggregated tally.

In this section, we present various election results functions f^{res}, explaining unique properties, for example, if and how the result function differs from the above scheme.

2.7.1. Aggregated Tally

The typical election result function used in practice is f_{AggTally}. Using the aggregated tally as input, this function is the identity: the election result function takes as input the aggregated tally $n_{\mathsf{votes}}^1, \ldots, n_{\mathsf{votes}}^{n_{\mathsf{cand}}}$ and outputs the aggregated tally $n_{\mathsf{votes}}^1, \ldots, n_{\mathsf{votes}}^{n_{\mathsf{cand}}}$, i.e., it outputs for every candidate how many votes this candidate received. However, the aggregated tally is typically not the election result function of interest. However, based on the result on f_{AggTally}, everyone can compute the actual voting result, for example, which candidate received the most votes.

2.7.2. Plurality Voting

In *plurality voting* (or *first-past-the-post voting*), the election winners are the candidates with the most votes, regardless of whether they secure an absolute majority (more than 50% of the total number of votes) or not. Plurality voting is a straightforward and widely used method in many elections. The principle of plurality voting is that the candidate with the most individual support should be declared the winner, even if it is not a majority. While simplicity and ease of understanding are advantages of this method, it can lead to outcomes where the winner does not necessarily represent the majority's preferences or enjoy broad support. This phenomenon is particularly evident in cases with multiple candidates, where the winning candidate may have received fewer votes than the combined total of other candidates.

The election result function $f^{\text{res}}_{\mathsf{Plurality}}$ takes as input the aggregated tally $n^1_{\mathsf{votes}}, \ldots, n^{n_{\mathsf{cand}}}_{\mathsf{votes}}$ and outputs a list of bits $(b_1, \ldots, b_{n_{\mathsf{cand}}}), b \in \{0,1\}$ such that $b_i = 1$ if (and only if) $n^i_{\mathsf{votes}} = \max_{i=1,\ldots,n_{\mathsf{cand}}} n^i_{\mathsf{votes}}$. This election result function outputs the candidates that received the most votes.

2.7.3. Threshold

We refer to election methods where every candidate surpasses a certain threshold of votes wins as *threshold-based* systems. In such systems, we establish a predetermined threshold, and any candidate who receives votes exceeding this threshold is declared a winner. We can set the threshold differently, such as a specific number of votes or a percentage of the total votes cast. Typically, proportional representation systems employ this method, where the goal is to ensure fair representation for various parties or groups. By allowing candidates who meet or exceed the threshold to secure seats or positions, this method allocates representation proportionately to the support received. Using a threshold-based system promotes inclusively and provides opportunities for smaller parties or candidates to gain representation, fostering a diverse and representative political landscape.

The election result function $f^{\text{res}}_{\mathsf{Threshold},t}$ uses as parameter a number t. This result function takes as input the aggregated tally $n^1_{\mathsf{votes}}, \ldots, n^{n_{\mathsf{cand}}}_{\mathsf{votes}}$ and outputs a list of bits $(b_1, \ldots, b_{n_{\mathsf{cand}}}), b \in \{0,1\}$ such that $b_i = 1$ if (and only if) $n^i_{\mathsf{votes}} \geq t$. This election result function outputs all candidates with at least t many votes.

2.7.4. Ranking

The election result function $f^{\text{res}}_{\mathsf{Ranking}} : (\mathbb{N})^{n_{\mathsf{cand}}}_{i=1} \to \{0, \ldots, n_{\mathsf{cand}} - 1\}^{n_{\mathsf{cand}}}$ takes as input the aggregated tally $n^1_{\mathsf{votes}}, \ldots, n^{n_{\mathsf{cand}}}_{\mathsf{votes}}$ and outputs a list of ranks $(r_1, \ldots, r_{n_{\mathsf{cand}}})$ such that each rank states against how many other candidates a candidate wins or ties a direct comparison. That is, $r_i = k$ means that there are k other candidates that received less or equal many votes than c^{cand}_i. This method reveals more information than $f^{\text{res}}_{\mathsf{Plurality}}$.

2.7.5. Partial Ranking

While $f^{\text{res}}_{\mathsf{Ranking}}$ outputs the ranking of all candidates, we can also output only a partial ranking. We only output the ranks of the n best candidates. The election result function $f^{\text{res}}_{\mathsf{PartialRanking},n} : (\mathbb{N})^{n_{\mathsf{cand}}}_{i=1} \to \{0, n_{\mathsf{cand}} - n, \ldots, n_{\mathsf{cand}} - 1\}^{n_{\mathsf{cand}}}$ takes as input the aggregated tally and outputs a list of ranks $(r_1, \ldots, r_{n_{\mathsf{cand}}})$, where the function distributes the ranks $n_{\mathsf{cand}} - 1$ to $n_{\mathsf{cand}} - n$ and sets the ranks of all remaining candidates to zero.

2.7.6. Best

The election result function $f^{\text{res}}_{\mathsf{Best},n}$ represents the *plurality-at-large* or *block voting* method. In this electoral system, a predetermined number, n, of candidates with the highest number of votes

Candidate	Grade				
	A	B	C	D	E
c_1^{cand}	0	12	0	0	11
c_2^{cand}	10	0	2	0	10

Table 2.2.: Example of an aggregated tally for $f^{\mathrm{res}}_{\mathsf{MJMedian}}$.

are declared winners. Elections in multi-seat constituencies or elections for multiple positions often employ this method. The candidates who secure the highest individual vote counts, up to the specified number of winners, are elected. However, this method can result in a lack of proportional representation, as it may not accurately reflect the overall preferences of the electorate.

The election result function $f^{\mathrm{res}}_{\mathsf{Best},n} : (\mathbb{N})_{i=1}^{n_{\mathsf{cand}}} \to \{0,1\}^{n_{\mathsf{cand}}}$ takes as input the aggregated tally $n^1_{\mathsf{votes}}, \ldots, n^{n_{\mathsf{cand}}}_{\mathsf{votes}}$ and outputs a list of bits $(b_1, \ldots, b_{n_{\mathsf{cand}}}), b \in \{0,1\}$ such that $b_i = 1$ if (and only if) the rank of c_i^{cand} is at least $n_{\mathsf{cand}} - n$.

In other words, this election result function outputs a partial ranking $f^{\mathrm{res}}_{\mathsf{PartialRanking},n}$ but hides the individual ranks.

2.7.7. Ranking with Aggregated Votes of Best

There exist arbitrary combinations on which rankings to reveal. The Deutsche Forschungsgesellschaft (DFG) uses a variant for the elections of the Fachkollegien (review boards) that works as follows. The election result function $f^{\mathrm{res}}_{\mathsf{RankingVotesBest},n}$, parameterized by a value n, outputs the ranking of all candidates and the number of votes each of the best n candidates have received.

2.7.8. Majority Judgment

Majority judgment aims to go beyond a simple tally of votes by considering the quality or acceptability of each candidate according to the judgments voters assign. There are multiple possible procedures to evaluate the election result. In the following, we present two variants.

2.7.8.1. Median Grade

This majority judgment variant calculates and outputs each candidate's average judgment. That is, $f^{\mathrm{res}}_{\mathsf{MJMedian}}$ computes the median grade per candidate.

For example, consider the candidates $c_1^{\mathsf{cand}}, c_2^{\mathsf{cand}}$ and the set of grades A, B, C, D, E, where the grades are ordered $A > B > C > D > E$. Table 2.2 shows an example of a possible aggregated tally, for instance, c_1^{cand} receives the grade B 12 times and the grade E 11 times. Using Table 2.2, the median grade of c_1^{cand} is B and the median grade of c_2^{cand} is C. Thus, the result is (B, C).

2.7.8.2. Full Majority Judgment

One can find the candidate with the best median grade to get one winning candidate. If no such unique candidate exists, this evaluation method (what we will call *full majority judgment*, denoted as $f^{\text{res}}_{\mathsf{MJFull}}$) uses multiple rounds. In each round, the best obtained median grade is computed. It eliminates all candidates that currently have a worse median grade. Then, it removes one vote from this median grade for every remaining candidate. For the next round, it updates the median grades accordingly to the removal of the single vote. The method repeats this procedure until only one candidate, the election winner, is left.

The algorithm described above is highly inefficient because the number of iterations scales in the number of voters (it removes a single vote in each iteration step). Balinski et al. [BL07] describe an alternative algorithm that only scales in the number of candidates and grades, not the number of voters. The algorithm still computes the same winner as the algorithm above does. Even more, this algorithm allows for aggregation of the votes by using $\mathfrak{C}_{\mathsf{MajorityJudgment}}$. In this choice space, the ballots are a matrix of the candidates and grades, where entry A_{ij} is set to one if candidates i receive grade j, and zero otherwise. Then, in each iteration, the algorithm eliminates all votes to the median grade according to $f^{\text{res}}_{\mathsf{MJMedian}}$ and then computes the new median grade.

2.7.9. Condorcet Methods

Condorcet methods are voting methods that (initially) share the aim of determining a so-called *Condorcet winner*, i.e., a candidate that would beat all other candidates in a runoff voting between them. That is, c_i^{cand} is the Condorcet winner if $\forall j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\} : n^i_{\mathsf{votes}} > n^j_{\mathsf{votes}}$. We will call these pairwise runoffs *duels*.

However, there are cases where no Condorcet winner can be determined. This situation is known as a *Condorcet paradox* or *Condorcet cycle*. It occurs when no candidate can win against every other candidate in pairwise comparisons. In such cases, additional methods may be employed to determine the winner, such as applying specific tie-breaking rules. Hence, several variations loosen this aim to make it more likely to output a winner. Namely, in these variations, the election result usually consists of the Condorcet winner if such a candidate exists. However, they can still generate an election result if no Condorcet winner exists.

Condorcet voting is a more comprehensive method than other voting systems like plurality voting, as it considers the relative preferences of voters for all candidates rather than just the top choice.

The voters cast as ballot a $n_{\mathsf{cand}} \times n_{\mathsf{cand}}$ ranking matrix ψ (see Section 2.6.3.2), which denotes in each entry ψ_{ij} whether the voter prefers c_i^{cand} over c_j^{cand} ($\psi_{ij} = 1$ if that is the case, $\psi_{ij} = 0$ otherwise). As input for the Condorcet election result functions, we use the aggregated ranking matrix Ψ, which is an entry-wise addition of all voters' duel matrices, i.e., Ψ_{ij} denotes how often candidate c_i^{cand} was preferred over candidate c_j^{cand}.

In the following, we present various Condorcet variants.

2.7.9.1. Plain Condorcet

The vanilla/plain Condorcet method (denoted as $f^{\text{res}}_{\mathsf{CondorcetPlain}}$) outputs a single Condorcet winner, i.e., a single candidate that wins all duels. This result function takes as input the aggregated duel matrix $\mathsf{DM}^{\mathsf{agg}}$ and outputs a list of bits $(b_1, \ldots, b_{n_{\mathsf{cand}}}), b \in \{0, 1\}$ such that $b = 1$ if (and only if) c_i^{cand} is the Condorcet winner, i.e. it holds that $\forall j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\} : \Psi_{ij} > \Psi_{ji}$. This method is not guaranteed to output a result. For example, consider three candidates $c_1^{\mathsf{cand}}, c_2^{\mathsf{cand}}$, and c_3^{cand} such that c_1^{cand} wins the direct comparison against c_2^{cand}, c_2^{cand} wins the direct comparison against c_3^{cand}, and c_3^{cand} wins the direct comparison against c_1^{cand}. Then, no single candidate wins against each other; thus, no single Condorcet winner exists.

2.7.9.2. Weak Condorcet

In this method $f^{\text{res}}_{\mathsf{CondorcetWeak}}$, all candidates that do not lose duels are output. That is, c_i^{cand} is a weak Condorcet winner if it holds true that $\forall j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\} : \Psi_{ij} \geq \Psi_{ji}$. In contrast to the plain Condorcet method, the weak Condorcet method can output multiple candidates. Such outputs can happen if multiple candidates are tied and win against every other candidate. If there is a (necessarily unique) Condorcet winner, this method will only output the Condorcet winner. As plain Condorcet, this method does not guarantee winning candidates since each candidate could lose some duel. This method is also not guaranteed to output winner(s); applying the example presented for plain Condorcet, no candidate does not lose any duel.

2.7.9.3. Copeland

This method $f^{\text{res}}_{\mathsf{CondorcetCopeland}}$, as opposed to the previous two methods, is guaranteed to output some winning candidate(s). To do so, it considers the wins and losses of each candidate in their duels and outputs all candidates with the highest difference between wins and losses. A Condorcet winner – if existent – would reach the highest possible difference; hence, the Copeland method will output the Condorcet winner if such a candidate exists. Using the example from above, the Copeland method outputs the set $\{c_1^{\mathsf{cand}}, c_2^{\mathsf{cand}}, c_3^{\mathsf{cand}}\}$ as the election result.

2.7.9.4. Smith Set

In this method $f^{\text{res}}_{\mathsf{CondorcetSmithSet}}$, a set of winners is output, the so-called *Smith set*. The Smith set is the set of candidates of minimal size such that each candidate from the Smith set wins the duels against every candidate outside the Smith set. If a Condorcet winner exists, the Smith set will consist of this candidate. Otherwise, it will necessarily contain multiple candidates.

2.7.9.5. Minimax

The Minimax method $f^{\text{res}}_{\mathsf{CondorcetMiniMax},\kappa}$ uses a metric κ, and the idea of this method is only to consider the *worst* duel of each candidate and then declare all candidates that have the *best* of these worst duels as winners. For this purpose, one needs to define some metric κ to assign

scores $\theta \in \{0, \ldots, n_{\mathsf{votes}}\}$ to the duels, i.e., the *worst* (or *best*) duel is the one with the lowest (or highest) score. The selection of this metric κ defines a concrete instantiation of the minimax method. The minimax election result function computes for each candidate the score of the *worst* duel (according to the given metric) and then finds the candidate with the *best worst* score. Here, we present the minimax method with two metrics, namely *margins* and *winning votes*. Minimax does output a Condorcet winner for these two scores if it exists. However, using other score functions, this is not necessarily the case.

Minimax Margins. In the case of *margins*, denoted by MarginMetric, the score of the candidate c_i^{cand}'s duel versus candidate c_j^{cand} is the difference between the number of duels won by c_i^{cand} versus c_j^{cand} minus the number of duels lost by c_i^{cand} versus c_j^{cand}.

Minimax Winning Votes. In the case of *winning votes*, denoted by WinningVotesMetric, the score of c_i^{cand}'s duel versus c_j^{cand} is 0 if c_i^{cand} wins more duels versus c_j^{cand} than c_i^{cand} loses. Otherwise, it is given by the negative number of won duels of c_j^{cand} versus c_i^{cand}.

2.7.9.6. Schulze

The Schulze method $f^{\mathrm{res}}_{\mathsf{CondorcetSchulze}}$ is a slightly more complicated but commonly used Condorcet method (see Section 2.8). In this method, we also consider a score function for the duels. Since, in practice, elections use the score function of the Copeland method, we will use this function in this thesis. We represent the candidates and the duels between them as a complete directed weighted graph Γ, where the nodes of Γ represent the candidates and an arrow $c_i^{\mathsf{cand}} \to c_j^{\mathsf{cand}}$ is weighted with the score of c_i^{cand}'s duel versus c_j^{cand}. Now, for any path p in Γ, we define the *value* of p as the lowest weight among the arrows involved in p. We then consider the *path value matrix* Ω, an $(n_{\mathsf{cand}} \times n_{\mathsf{cand}})$-matrix with entry Ω_{ij} being the highest path value among paths from c_i^{cand} to c_j^{cand}. The Schulze method then outputs all candidates c_i^{cand} such that $\Omega_{ij} \geq \Omega_{ji}$ for each $j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\}$. This algorithm is called *Floyd-Warshall algorithm* [Hou10]. The Schulze method guarantees to output some candidate(s). With the metrics described in the Copeland method, this candidate is the unique Condorcet winner, if existent.

We present an example of the Schulze evaluation in Figure 2.2. For this example, we consider the candidates A, B, C, and D with the following aggregated tally. The ranking (A, C, D, B) (ordered from first to last preference) received two votes, the ranking (B, C, D, A) received three votes, the ranking (C, B, A, D) received seven votes, the ranking (C, D, A, B) received four votes, the ranking (D, B, A, C), and all remaining rankings received no votes. For this example, the set of winners is the set containing C, since only for this candidate it holds that $\Omega_{Cj} \geq \Omega_{jC}$ for all candidates.

2.7.10. Instant-Runoff Voting (IRV)

Instant-runoff voting (IRV) is a complex, often-used ranked voting method that runs over multiple rounds. It is, for example, used in the elections for the News South Wales Legislative Assembly in Australia [NSW20], India [Gov20], the UK [The11], and for governor and congress

(a) Directed weighted Schulze graph Γ. (b) Graph of the path value matrix Ω.

Figure 2.2.: Example of a Schulze evaluation.

elections in the state of Maine in the US [Mai20] (see Section 2.8). For IRV, the voters rank the candidates. Then, we evaluate one or more rounds until we find the election's winner. With $f^{\text{res}}_{\mathsf{IRVRoundTally}}$, we denote the function that takes as input the current aggregated tally and eliminated candidates and outputs how many first preferences each remaining candidate currently receives, and $f^{\text{res}}_{\mathsf{IRVRoundEliminated}}$ denotes the function that, on the same input, outputs the candidate to eliminate in the current round. In each round of the evaluation, we eliminate the candidate with the smallest number of first preferences. This process repeats until one of the candidates has received the absolute majority of votes in a round (which is the case if only one candidate remains). In the most common setting, only two rounds, and the two leading candidates of the first round proceed to the second round. This particular case is also known as *runoff voting*. An example of this is the French presidential elections [Ely12].

We differentiate between two types of ballots: In some cases, the voters must rank all candidates, casting a complete ranking as a ballot. Some IRV instantiations (for example, the election for the New South Wales Legislative Assembly [NSW20]) allow the voters to cast ballots with partial rankings.

If a candidate is ranked first by an absolute majority of voters, this candidate immediately wins the election. Otherwise, the evaluation function eliminates the candidate who has received the lowest first rankings, i.e., removed from the pool of candidates. Then, all votes that ranked an eliminated candidate first are redistributed and now counted according to the non-eliminated candidate they ranked highest. If a vote at some point only consists of ranks assigned to already eliminated candidates, the counting procedure will no longer consider this vote. This process repeats until one of the remaining candidates has received the majority of votes and thus wins the election.

An intrinsic issue regarding this voting method is how to deal with ties. More precisely, we need some regulation on which candidate to eliminate in any intermediate round if two (or more) candidates have the same first rankings. This issue has been addressed differently in various instantiations of IRV. For example, in the IRV method used in Maine [Mai20], ties are by default broken by lot. A random candidate is chosen and eliminated from the tied candidates. We denote this variant of IRV as $f^{\text{res}}_{\mathsf{IRVLotComplete}}$ with a complete ranking as input and $f^{\text{res}}_{\mathsf{IRVLotPartial}}$

with a partial ranking as input. Under certain circumstances, however, the electoral law of Maine allows for eliminating multiple tied candidates at once. The New South Wales Legislative Assembly election uses a more complex method of breaking ties than eliminating a random tied candidate, as explained in Section 2.8.

2.7.11. Parliamentary Elections with Party-Based Seat Allocation

In practice, a fundamental class of elections is *parliamentary elections with party-based seat allocation* as carried out by many countries worldwide. These are among the most complex types of elections: They usually involve millions of voters, dozens of parties, hundreds of individual candidates, and hundreds of electoral constituencies. In some cases, voters have not just one but multiple votes that they can distribute among parties and possibly also individual candidates. Sophisticated multi-step algorithms compute the election result, i.e., assigning seats to individual candidates. An essential component for this process is a so-called *seat allocation method*, which inputs the number of available seats and a set of parties with their total number of votes and then computes the number of seats assigned to each party.

Mixed electoral systems are voting methods that combine plurality or majority voting and proportional representation to combine the advantages of both methods. Usually, in a mixed electoral system, a voter has two votes - one for proportionality and one for such a system's plurality/majority aspect.

Typical examples of mixed electoral systems, such as the ones used in the German federal elections [Bun20], cover the proportional aspect of such a mixed system by a proportional party list voting, while the plurality/majority aspect is (for example) covered by first-past-the-post-voting among the parties' candidates from respective electoral districts.

The two main classes of mixed electoral systems are *non-compensatory* and *compensatory* systems. Non-compensatory systems treat the results from the proportional and plurality/majority aspects independently. For example, one could imagine an election to fill 200 seats - 100 through the proportional component and 100 via the majority/plurality component. In non-compensatory methods, the complete distribution of the 200 seats usually does not represent the proportions of the proportional component anymore. Using two independent votes is also known as *parallel voting*. Ukraine, Japan, and Lithuania (see [BG13]) use this variant of a mixed electoral system.

In contrast, *compensatory* systems first count the votes from the majority/plurality components and then award the seat(s) to the respective winning representative(s). Next, *compensatory* systems award further seats to the respective parties to recover the proportions from the proportional aspect. Hence, in a way, the proportional component is used to *compensate* any imbalances from the majority/plurality component.

Instantiations of non-compensatory systems mainly differ in the concrete choice of voting method for the majority/plurality component and the method used for allocating seats proportionally in the proportional component. A famous class of instantiations called *mixed-member proportional representation (MMPR)* is used, for example, for the federal elections and the

election of many state parliaments in Germany [Bun20], the elections for the Scottish and Welsh parliaments (see [UK 21], note that in the UK the term *additional member system* is used to describe MMPR) and the elections for the New Zealand House of Representatives [MoJ93]. The majority/plurality component is typically instantiated here via first-pass-the-post-voting, where a single representative from the respective electoral district is elected. For the proportional component, there are several methods of allocating seats. In the following, we present two of them, namely the *Hare-Niemeyer method*, used, e.g., in Ukraine and Italy, and the *Sainte-Laguë method*, used, e.g., in Indonesia and Germany.

2.7.11.1. Hare-Niemeyer Method

Many elections use the *Hare-Niemeyer method*, e.g., parliamentary elections in Ukraine and Italy, and also the German federal elections used this method until 2005 [Deu]. The Hare-Niemeyer method works as follows: Assume we assign n_{seats}. Then, if there are a total of v_{total} valid votes and party c_i^{party} has received n_{votes}^i votes, we compute the number of seats we award to c_i^{party} using the *ideal quota* given by

$$q_i := \frac{n_{\mathsf{votes}}^i \cdot n_{\mathsf{seats}}}{v_{\mathsf{total}}}.$$

We award c_i^{party} initially $\tilde{s}_i := \lfloor q_i \rfloor$ seats. However, since these $\tilde{s}_i$ usually do not add up to n_{seats}, we distribute the remaining $k \in \{1, \ldots, n_{\mathsf{parties}} - 1\}$ seats in order of the highest remainders of $\frac{n_{\mathsf{votes}}^i \cdot n_{\mathsf{seats}}}{v_{\mathsf{total}}}$. The k parties with the highest decimal places $q_i - \tilde{s}_i$ receive an additional seat.

2.7.11.2. Sainte-Laguë

The *Sainte-Laguë method* (also called *Webster method*) is a seat allocation method, i.e., a procedure that describes how to allocate a given number of seats to a set of parties depending on the number of votes each party has received. The Sainte-Laguë method is used by parliamentary elections in many countries, for example, Indonesia, New Zealand, Nepal, Sweden, Norway, Germany, and Kosovo. As part of computing the election result, these elections run the Sainte-Laguë method multiple times on different inputs. For example, the official evaluation of the final seat distribution of the German Bundestag of the election in 2021 required to run the Sainte-Laguë method 23 times (in addition to several other steps, as explained in Section 2.8).

There are essentially two distinct (but provably equivalent [Lij83]) algorithms for computing the seat allocation following the Sainte-Laguë method, one based on highest quotients and one on finding suitable denominators. Both algorithms take as input the number of seats n_{seats} and, for each party $j \in \{1, \ldots, n_{\mathsf{parties}}\}$, the total number of votes n_{votes}^j that party j has received. They return the number of seats assigned to each party.

- *Highest-Quotients.* For $i \in \{1, \ldots, n_{\mathsf{seats}}\}, j \in \{1, \ldots, n_{\mathsf{parties}}\}$ compute the quotients $q_i^j := \frac{n_{\mathsf{votes}}^j}{2(i-1)+1}$. Let M be the list of the n_{seats} highest quotients. Then party j is assigned k seats, where k is the number of quotients in M that belong to j, i.e., quotients of the form $q_i^j, i \in \{1, \ldots, n_{\mathsf{seats}}\}$.
- *Suitable-Denominator.* Given a *suitable denominator* d, party j receives $n_{\mathsf{seats}}^j = \lfloor \frac{n_{\mathsf{votes}}^j}{d} \rceil$ seats, where $\lfloor \cdot \rceil$ denotes rounding to the closest integer (rounding of .5 can be chosen to be either up or down and can be chosen differently for each j). A denominator d is *suitable* if the result of this computation leads to the number of desired total seats, i.e., if $\sum_j n_{\mathsf{seats}}^j = n_{\mathsf{seats}}$. To find a suitable denominator, one generally starts with an arbitrary denominator d, e.g., $d = \left\lfloor \frac{\sum_j n_{\mathsf{votes}}^j}{n_{\mathsf{seats}}} \right\rceil$, checks the corresponding number of seats that would be assigned, and then tweaks d until finding a suitable value.

Ties can occur in both of these algorithms. In the highest-quotients algorithm, there might be two equal quotients, but only sufficient seats are available to distribute them. In the suitable-denominator algorithm, all suitable denominators may be such that the quotients of multiple parties end on .5, some rounded up, and others rounded down to achieve an overall sum of n_{seats}. The Sainte-Laguë method does not define any specific tie-breaking mechanism. Instead, elections using this method additionally need to specify how they handle ties.

2.8. Real-World Elections

This section presents real-world elections that use the above-presented voting methods and election result functions. In this thesis, we will construct e-voting systems for these elections. We provide the instantiation $(\mathfrak{C}, f^{\mathrm{res}})$ used to perform real-world elections.

2.8.1. House of Commons

The House of Commons is the lower house of the Parliament of the United Kingdom, and its 650 members are elected through a general election held every five years. However, the Prime Minister has the power to call an early election if they have the support of two-thirds of the House of Commons or if a vote of no confidence is passed against the government. The electoral system used in the House of Commons is known as *first-past-the-post-voting*. Therefore, each of the 650 geographical constituencies in the UK elects one Member of Parliament (MP). The candidate who receives the highest number of votes in a constituency becomes the representative for that area in the House of Commons. Political parties and independent candidates nominate individuals to stand as candidates in each constituency. Parties usually select candidates through a selection process, such as primary elections or party members' votes.

On election day, eligible voters in each constituency cast their votes at designated polling stations. Voters mark their preferred candidate on the ballot paper (using $\mathfrak{C}_{\mathsf{Single}}$). After the polls close, local authorities count the votes in each constituency. The candidate who receives

the highest number of votes in a constituency is declared the winner (ties are broken by lot) and becomes the Member of Parliament for that area. The results from all constituencies are collected to determine the overall outcome of the election. The political party that wins the majority of seats in the House of Commons forms the government. If no party secures a majority, it results in a hung parliament. Parties may form coalitions or alliances to establish a working majority and create a government. Once elected, MPs represent their constituencies in the House of Commons. They participate in debates, vote on legislation, scrutinize the government, and represent the interests and concerns of their constituents.

Formally, this election uses $\mathfrak{C}_{\mathsf{Single}}$ and $f^{\mathrm{res}}_{\mathsf{Plurality}}$ in each of the 650 constituencies with tie-breaking by lot. In the general elections in 2019, no constituency had more than $77,062$ eligible voters.

2.8.2. Elections for the Fachkollegien in the Deutsche Forschungsgesellschaft

The Deutsche Forschungsgesellschaft (DFG) runs elections for their Fachkollegien (review boards) [Deu19]. The elections in 2019 had 410 Fachkollegien, each elected using $f^{\mathrm{res}}_{\mathsf{RankingVotesBest},n}$ with around $1,000$ voters for each election and at most 32 candidates (*FK 201 Grundlagen der Biologie und Medizin*).

2.8.3. Grand Final of the Eurovision Song Contest

The Eurovision Song Contest (ESC) [Eur20] is an annual international music competition where participating European countries (and some from outside Europe) send representatives to perform original songs. The process of selecting the winner of the Grand Final of the ESC involves several stages:

1. Each participating country holds its national selection process to choose its representative and song for the ESC. The methods for selecting the representative can vary, ranging from public televoting to expert juries or a combination of both.

2. The ESC typically includes two semi-finals before the Grand Final. These semi-finals serve as a place in the final. The *Big Five* countries (France, Germany, Italy, Spain, and the United Kingdom) and the host country (the previous year's winner) automatically qualify for the Grand Final. Thus, they do not compete in the semi-finals. The number of countries that advance to the qualifying rounds, where the participating countries compete for a final from each semi-final, is determined by a combination of televoting and jury voting. The best ten counties of each semi-final advance to the Grand Final of the ESC.

3. In the Grand Final, viewers from each participating country can vote for their favorite songs through televoting or SMS. Additionally, each country has a jury panel of music industry professionals who provide their rankings and votes. The televoting results and jury votes are combined to determine the final scores for each participating country.

During the voting segment of the Grand Final, spokespersons from each country announce the points awarded by their country's jury and televoting results. The points are on a 12-point scale, with countries awarding points from the list $\mathcal{P}_{\mathsf{ESC}} := (12, 10, 8, 7, 6, 5, 4, 3, 2, 1)$ to their top ten favorite songs, both from the jury and televoting, where no points may be awarded for the entry of their own country. The combined scores from all participating countries are then tallied. The winner of the ESC is the country with the highest combined score from the jury and televoting. The winner's announcement usually creates suspense as the broadcast gradually reveals the points.

If two or more participants end up with the same number of points, further differentiation criteria apply to ensure a unique ranking. First, the number of countries from which the respective participants have received points is decisive. If this does not allow a unique ranking, the number of maximum scores awarded to the respective participants is considered. In this case, the number of 12-point scores is evaluated first, followed by the number of 10-point scores in the event of a tie, and so on. If there is no difference by comparing all the individual scores, the countries concerned will be placed according to the order of the starting numbers.

Formally, the ESC uses the choice space $\mathfrak{C}_{\mathsf{ESC}} = \mathfrak{C}_{\mathsf{BordaPointList}}(\mathcal{P}_{\mathsf{ESC}})$ with the restriction that if the voter is also a candidate, she may not assign points to herself. The election result function is $f^{\mathrm{res}}_{\mathsf{Plurality}}$, without considering the above tie-breaking mechanism. There are 26 candidates (the *Big Five* countries, the country that won the ESC last year, and ten countries each from the two semi-finals). In 2023, there were 37 countries that participated in the voting, and thus $n_{\mathsf{voters}} = 37$.

2.8.4. Parliamentary Elections in the Republic of Nauru

Borda voting is a prominent ranked voting method, which is famously used for national elections, e.g., for parliamentary elections in the Republic of Nauru [Rep16].

Nauru follows a unicameral system with a single legislative body called the Parliament. The Parliament consists of 19 members who are elected from eight constituencies by the citizens of Nauru. Eligible voters in Nauru, typically Nauruan citizens aged 20 years or older, have the right to participate in the parliamentary elections. However, specific requirements and eligibility criteria may apply. Before the election, individuals interested in running for Parliament can submit their nominations within a specified time frame. They may need to fulfill specific qualifications, such as being a citizen of Nauru or meeting residency requirements. Once the candidates are nominated and approved, they can campaign to seek support from the voters. Registered voters go to their designated polling stations on election day to vote. The electoral authorities of Nauru can determine the exact voting procedures, including using voting booths or electronic voting machines. After the voting concludes, the ballots are collected, and the vote-counting process begins. The electoral authorities count the votes, and the candidates who receive the highest number of votes are declared the winners.

The election for the parliament of Nauru uses the *Dowdall system*, a special case of Borda voting: The voter ranks the candidates according to the point list $\mathcal{P}_{\mathsf{Dowdall}} := \{1, \frac{1}{2}, \frac{1}{3}, \ldots, \frac{1}{n_{\mathsf{cand}}}\}$. The election is then defined by the choice space $\mathfrak{C}_{\mathsf{Dowdall}} = \mathfrak{C}_{\mathsf{BordaPointList}}(\mathcal{P}_{\mathsf{Dowdall}})$ and $f^{\mathrm{res}}_{\mathsf{Best},19}$. In the elections of 2022, Ubenide, the largest of the eight constituencies, had 18 candidates and 1,630 voters.

2.8.5. Debian Project

The Debian Project, a renowned open-source, community-driven initiative, utilizes elections as an integral part of its governance structure. Debian is a widely popular operating system and software ecosystem developed and maintained by a global community of volunteers. The project organizes elections to choose individuals for leadership roles, such as the Debian Project Leader (DPL). The DPL is responsible for representing the project, coordinating with developers, making important decisions, and maintaining the project's overall direction. Through these elections, Debian ensures that its leadership positions are filled through a democratic process, allowing community members to have a say in shaping the project's future and ensuring accountability and transparency in its governance. The Debian Project uses $\mathfrak{C}_{\mathsf{RankingPermutation}}$ as choice space and Condorcet Schulze $f^{\mathrm{res}}_{\mathsf{CondorcetSchulze}}$ as election result function [Deb12]. The elections of the DPL use this method. The elections in 2023 had 2 candidates, while the elections in 2022 had 4 candidates.

Nevertheless, other elections also use this method. For example, in 2022, a general resolution regarding non-free firmware was elected. This election consisted of 7 choices. Elections in the Debian Project usually consist of about 300 voters.

2.8.6. The New South Wales Legislative Assembly

The New South Wales Legislative Assembly is the lower house of the Parliament of New South Wales, Australia [NSW20]. Elections for the Legislative Assembly are held every four years. The voting method used in this election allows voters to rank candidates in order of preference, but they also have the option to vote for only one candidate without ranking the rest. Therefore, we model this voting method with partial ranking permutations, which are represented using $\mathfrak{C}_{\mathsf{Single}}$ (see Section 2.6). New South Wales is divided into electoral districts, also known as electorates, which a single Member of the Legislative Assembly represents. There are 93 electorates in total, and each electorate elects one member. Political parties and independent candidates nominate individuals to stand as candidates in each electorate. Parties usually select candidates through internal processes, such as pre-selection, while independent candidates can nominate themselves. On election day, eligible voters in each electorate cast their votes at designated polling stations. Voters mark their preferred candidate by placing a number "1" next to their chosen candidate and can continue to rank other candidates if they wish. Alternatively, they can vote for one candidate without ranking the rest. Tallying the votes is done using IRV. However, with a sophisticated tie-breaking mechanism, if there is a tie between two (or more) candidates for

elimination in any evaluation round r, the candidates are compared based on the ballots from round $r-1$. The election process eliminates the candidate ranked first by the least votes in that round. If there is still a tie between the candidates in that round, the process is repeated with round $r-2$, iterating through all previous rounds. In rare cases where the tie cannot be resolved this way because there is a tie between the candidates in all previous rounds, a lot decides who is eliminated. We denote this variant of IRV as $f^{\text{res}}_{\mathsf{IRVNSWCompelte}}$ with a complete ranking as input and $f^{\text{res}}_{\mathsf{IRVLotPartial}}$ with a partial ranking as input.

The number of candidates per electorate varies between 5 (e.g., the electorates of Newtown, Tockdale, Terrigal, Winston Hills) and 10 (the electorate of Murray), with about $55,000$ voters per electorate.

2.8.7. The Maine House of Representatives

The Maine House of Representatives in the USA consists of 155 members plus three additional seats for local minorities, elected for two years [Mai20]. The elections consist of 155 districts with one victor each, determined by IRV. Tied candidates are eliminated by lot, and mathematically impossible candidates (as defined in [Sta19]) are eliminated in each round. This procedure employs $\mathfrak{C}_{\mathsf{Single}}$ and $f^{\text{res}}_{\mathsf{IRVLotPartial}}$ (see Section 2.6). Each district has approximately $8,000$ eligible voters and three candidates.

In Maine, voters elect the US President and Senator, with about five nominees for the latter. In 2020, there were six USA Presidential contenders with around 800,000 eligible voters.

2.8.8. Bundestagswahl (Germany)

Complex real-world elections for which we construct secure tally-hiding e-voting systems are the elections for the German Bundestag. This section explains these elections.

The German people directly elect the German federal parliament in an election combining proportional representation and first-past-the-post voting. This large-scale election uses multiple steps to determine how many seats the parliament will have (the initial number of 598 seats increases to represent the proportion of the votes) and which candidates obtain a seat. The 2021 election of the German Bundestag had $61,181,072$ eligible voters and 47 parties with $6,211$ candidates, distributed over 299 constituencies.

Each voter has two votes: a constituency vote (called *first vote*) and a party list vote (called *second vote*), both using $\mathfrak{C}_{\mathsf{Single}}$. The first votes are individually evaluated for each of the 299 constituencies: The candidate with the most votes wins the constituency and a seat in the parliament. The remaining seats of the parliament are distributed between the 16 German states in proportion to the number of inhabitants. The second votes are then used to obtain a proportional number of seats per party, first on a national layer, then in the states. In more detail, the election for the German Bundestag works as follows.

Notation. Let L be the set of all German states. We denote with $s^{\mathrm{d}}_{j,l}$ the number of direct seats that party j received in state l, and with $s^{\mathrm{q}}_{j,l}$ we denote the number of quota seats of party j in state l.

Single-member constituency seats. In each constituency, the candidate that receives the most constituency votes (first votes) wins this constituency and obtains a direct seat in the Bundestag. Possible ties are broken by lot. The number of candidates per constituency varies between 7 and 18.

Determine which parties enter the Bundestag. A party enters the Bundestag if it gets at least 5% of the party list votes (second votes) or won at least 3 constituencies. An exception to this rule is parties representing minorities in Germany. They do not have to meet these criteria to enter the Bundestag [Bun23]. In the election of the German Bundestag in 2021, the *South Schleswig Voters' Association* (*Südschleswigscher Wählerverband*, SSW, in German) obtained one seat as a party representing a minority. Nevertheless, if a candidate wins a constituency, he is guaranteed a seat in the parliament, even if his party is not considered in the subsequent evaluation.

First top distribution: seat contingents of the states based on their inhabitants. The first distribution assigns the initial 598 seats of the Bundestag to the 16 German states such that each state has a seat contingent. This computation is done using an execution of the Sainte-Laguë method. This Sainte-Laguë method uses the number of inhabitants of each state as input.

First low distribution: distribution of the seat contingents to the parties' state lists. Knowing how many seats are assigned to each state, these seats are now distributed to the state's parties in proportion to the second votes in the state. A Sainte-Laguë method is executed for each state, with the number of second votes per party in this state as input. The number of seats that party j receives in the state by this method is called quota seats and denoted with s^{q}_{j} in the following. Typically, between 5 and 127 seats must be distributed between about 6 parties.

Minimal number of seats per party. Now, for each party j the minimum number of seats $s^{\min}_{j}$ that the party receives in the Bundestag is computed. For this, we compute $s^{\min}_{j,l}$, which denotes for party j and state l, how many seats party j receives in state l. This value is computed as follows:

$$s^{\min}_{j,l} := \max\left(\left\lceil \frac{s^{\mathrm{d}}_{j,l} + s^{\mathrm{q}}_{j,l}}{2} \right\rceil, s^{\mathrm{d}}_{j,l}\right)$$

Now, party j receives at minimum $s^{\min}_{j} = \sum_{l \in L} s^{\min}_{j,l}$ seats in the Bundestag. Additionally, each party has a threatening overhang: If party j won more direct seats $s^{\mathrm{d}}_{j,l}$ in state l than quota seats $s^{\mathrm{q}}_{j,l}$, this party has a threatening overhang of the difference between these two values. The threatening overhang is zero if the party wins at least as many quota seats as direct seats. Threatening overhang seats denote direct seats that a party won in a state but are not covered by their quota seats. Awarding the party these seats breaks the proportional representation of

the voting result. Therefore, the overall number of seats in the Bundestag is increased such that these overhang seats do not change the proportion of the votes too much.

Second top distribution: increasing the number of seats of the Bundestag. Now, a second top distribution is computed that determines how to handle the threatening overhang seats by increasing the number of seats in the Bundestag. On an intuitive level, in this step, the number of seats of the Bundestag is increased until the Sainte-Laguë method outputs a seat distribution in which every party j obtains their respective minimal number of seats $s_j^{\min}$ except three seats in total. During this step, ties are not broken by lot, but all parties in the tie obtain a seat. Therefore, more seats will be distributed among the parties in case of ties.

Moreover, in this step, we only consider parties that enter the Bundestag, i.e., parties that do not obtain any seat in the Bundestag are excluded from the following calculations since they do not influence the election result. We denote the set of parties that enter the Bundestag with P^{BT}. Instead of increasing the number of seats one by one until a suitable seat distribution is found, a more efficient algorithm is applied. That is, a suitable divisor for the Suitable-Denominator Sainte-Laguë method (see Section 2.7), that fulfills all requirements is determined. To find this divisor d, a divisor d_{no} that does not take overhang seats into account and a divisor d_{overh} that does take overhang seats into account is computed first:

- The divisor d_{no}, which does not take overhang seats into account, is determined as follows:

$$d_{\mathrm{no}} = \min_{j \in P^{\mathrm{BT}}} \left(\frac{v_j}{s_j^{\min} - 0.5} \right)$$

  where v_j denotes the number of second votes for party j on the national level.

- The divisor d_{overh}, which takes overhang seats into account, is determined as follows. If party j has threatening overhang, the set of possible overhang values D_{overh} is constructed as follows:

$$D_{\mathrm{overh}} = \left\{ \frac{v_j}{s_j^{\min} - i} \;\middle|\; i \in \{0.5, 1.5, 2.5, 3.5\}, j \in P^{\mathrm{BT}} \text{ and j has overhang seats} \right\}$$

  where v_j denotes the number of second votes for party j nationally. Note that 2.5 and 3.5 are only used if the party has 2 or 3 overhang seats. The formula includes four possible values since the first three overhang seats must not be balanced. The value d_{overh} is the fourth smallest element of D_{overh}.

The suitable divisor d is the minimum of both divisors: $d = \min(d_{\mathrm{no}}, d_{\mathrm{overh}})$. This divisor satisfies the requirements, ensuring that every party j obtains at least $s_j^{\min}$ seats in the parliament, except at most three.

Second low distribution. Executing the Sainte-Laguë method with the divisor d selected in the previous step might lead to a number of seats larger than the initial size of 598 seats of the Bundestag. As only the total seats for each party are calculated so far, these seats are distributed among the states. This is done by distributing the seats for each party j with the Sainte-Laguë method to the states using the number of second votes for j in each state as input.

Assigning overhang seats. The previous steps exclude up to three overhang seats. Thus, they must be assigned separately to suitable state lists of the parties. For each party j with such an overhang seat, the overhang set O^j_{overh} is computed as

$$O^j_{\text{overh}} = \left\{ \frac{v_{j,l}}{s^{\min}_{j,l} - i} \;\middle|\; i \in \{0.5, 1.5, 2.5\}, l \in L \right\},$$

where 1.5 and 2.5 are only used if the party has 2 or 3 overhang seats.

For n such overhang seat that party j receives, the minimal n values in O^j_{overh} are selected, and the corresponding states obtain the overhang seat.

Computing the final result. The resulting seat allocations are computed per party per state. First, the winning candidates of each constituency obtain a seat, and the remaining seats of the party in that state are assigned to the state list, starting from the top, excluding candidates that already have a seat.

2.9. Related Work

This section briefly presents related e-voting systems designed to be tally-hiding or protect against Italian attacks. We will discuss and compare them to our systems in the respective sections. Since we are the first to propose the concept of public tally-hiding and the first to construct such a (verifiable) system, no prior e-voting schemes follow this direction.

2.9.1. Fully Tally-Hiding E-Voting

In 1986, Benaloh proposed full tally-hiding and the first such system [Ben86]. Hevia and Kiwi tailored a tally-hiding e-voting system for jury voting [HK02]. Wen and Buckland showed how to perform tally-hiding for instant-runoff elections [WB09]. However, none of these protocols were formally proven secure or implemented to show practicality.

Szepieniec and Preneel proposed another fully tally-hiding e-voting protocol [SP15], but the system leaks information and is therefore not tally-hiding.

Canard et al. [CPST18] proposed a fully tally-hiding e-voting system for $f^{\text{res}}_{\mathsf{MJFull}}$. However, the underlying evaluation algorithm may not produce a result for every possible tally, which results in no verifiability guarantees. Furthermore, their implementation was not tested in a distributed network but on a single computer. Their protocol's performance in real-world distributed tallying scenarios is unclear due to the online complexity of the underlying MPC protocol.

Cortier et al. [CGY22] proposed fully tally-hiding MPC components for various election result functions, and they provide benchmarks for $f^{\text{res}}_{\mathsf{CondorcetSchulze}}$.

2.9.2. Partially Tally-Hiding E-Voting

Several protocols have been proposed to address the issue of Italian attacks in complex voting methods such as Condorcet, Borda, IRV, and single transferable vote (see, e.g., [CM05, Hea07, BMN+09, JRRS19, RCPT19]). All current systems mitigate Italian attacks, and some are efficient enough to be used in real-world elections. For instance, a protocol proposed by Ramchen et al. [RCPT19] was benchmarked using the 2015 New South Wales instant-runoff elections.

3. Full Tally-Hiding: Ordinos

Fully tally-hiding e-voting systems offer the highest level of confidentiality for the tally. These systems ensure that no party involved, including voters, election authorities, trustees, and even external observers, can gain any knowledge about the tally other than the election's result. Full tally-hiding ensures that the actual tally remains secret, and any intermediate values used for computing the election result are inaccessible to any party as long as there are sufficient many honest trustees. Even from the trustees who compute the election result, achieving this level of secrecy requires advanced heavy-weight cryptographic techniques like fully homomorphic encryption (FHE) and universally verifiable multi-party computation (MPC).

In our work of [KLM+20a], we propose Ordinos, the *first provable secure full tally-hiding e-voting* framework, which we instantiated for relatively simple single- and multi-vote elections.

In this chapter, we present the Ordinos framework. We then instantiate the Ordinos framework for further, more complex real-world elections. For this, we employ suitable cryptographic primitives, including an MPC protocol for greater-than tests. We implemented our resulting Ordinos instantiations and evaluated the performance, demonstrating the practicality of our instantiations.

We propose several innovative tally-hiding building blocks for the Ordinos framework, which we can utilize for parliamentary elections. Prior to our research, it was unclear whether and how we could perform parliamentary elections with party-based seat allocations in a tally-hiding manner. The Ordinos framework does not provide direct support for such elections, as constituencies, which are a crucial part of parliamentary elections, are not included. Hence, we modify the Ordinos framework and prove that the security properties of the original framework are still intact in our modified version. As a result, we propose the first verifiable and accountable tally-hiding voting system for a parliamentary election, specifically for the German parliament. Our findings from this study demonstrate, for the first time, that we can perform even a complex and large-scale real-world election in a verifiable and fully tally-hiding manner.

Our instantiations are integral for carrying out real-world elections using Ordinos. However, more is needed: additional aspects, such as client and verification interfaces, are out of scope. To bridge this gap, we offer a complete e-voting system that provides full tally-hiding and is ready for deployment in real-world elections. This web-based system supports all Ordinos instantiations except for the German Bundestag. The system comprises multiple components, each thoroughly documented and implemented, with graphical user interfaces accessible through the user's browser. Voters can cast their votes directly through their browser without downloading additional software. To perform Benaloh challenges [Ben07], we have developed an Android

application as a vote verification device. The verification process is fully automated and similar to that of [KMST16], and it is automatically triggered when the user opens the election result. This process happens in the user's browser, requiring no further user interaction.

Moreover, we provide a deeper understanding of tally-hiding in general, particularly in how far tally-hiding affects the level of ballot privacy of e-voting systems.

As explained in the introduction, this chapter covers the content and is based on the publications [KLM+20a, HHK+21a, WLH+23a, LAA+23b] and their corresponding technical reports [KLM+20b, HHK+21b, WLH+23b], and some parts of this chapter are taken verbatim from them.

We structure this chapter as follows. Section 3.1 presents the framework of full tally-hiding. In Section 3.2, we present Ordinos, the first provable secure fully tally-hiding e-voting system, including our instantiations of the Ordinos framework. We discuss related fully tally-hiding work in Section 3.3.

3.1. Fully Tally-Hiding Framework

We now introduce the notion of full tally-hiding e-voting. Intuitively, an e-voting protocol P is full tally-hiding for some voting method $(\mathfrak{C}, f^{\text{res}})$ if the following condition holds: Under the assumption that fewer trustees than a certain threshold t are dishonest, P provides the same level of privacy as the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\text{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ (which we will formally define in Section 3.1.1) for voting method $(\mathfrak{C}, f^{\text{res}})$, that reveals nothing but the actual election result by definition. That is, no one learns anything beyond the published election result. We will construct this ideal voting protocol in Section 3.1.1 and present and discuss the ideal privacy in Section 3.1.2. Based on these results, we evaluate the impacts of (full) tally-hiding in Section 3.1.3. Last, we formally define full tally-hiding in Section 3.1.4.

3.1.1. The Ideal Voting Protocol

In order to formally capture how the concept of tally-hiding affects elections, we first have to define an ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\text{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$. We present this ideal voting protocol in Figure 3.1. In this protocol, honest voters vote according to the vote distribution μ. Every run has $n_{\mathsf{voters}}^{\mathsf{honest}}$ many honest voters and n_{voters} voters overall. The ideal protocol collects the votes of the honest voters and the dishonest ones (where the latter ones are independent of the votes of the honest voters) and outputs the result according to the result function f^{res}. We note that the ideal voting protocol does not take the aggregated tally as input but all individual votes and computes f^{agg} as a first step before applying f^{res}.

3.1.2. Ideal Privacy

The privacy level δ (as defined in Definition 2.3) is larger than zero for virtually every voting protocol, as the result of the election always leaks some information. In order to have a lower

$\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\mathrm{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$

Parameters:

- Function $f^{\mathrm{res}}\colon \{0,1\}^* \to \{0,1\}^*$
- Probability distribution μ over $\mathfrak{C}$
- Numbers of voters n_{voters} and honest voters $n_{\mathsf{voters}}^{\mathsf{honest}}$
- $I \leftarrow \emptyset$ (initially)

On $(\mathsf{init}, \mathsf{honest})$ from S do:

1. $\forall i \in \{1, \ldots, n_{\mathsf{voters}}^{\mathsf{honest}}\}$: store $c_i^{\mathsf{choice}} \xleftarrow{\mu} \mathfrak{C}$
2. $I \leftarrow I \cup \{1, \ldots, n_{\mathsf{voters}}^{\mathsf{honest}}\}$
3. Return $\mathsf{success}$

On $(\mathsf{setChoice}, i, c^{\mathsf{choice}})$ from S do:

1. If $i \notin \{n_{\mathsf{voters}}^{\mathsf{honest}} + 1, \ldots, n_{\mathsf{voters}}\}$ or $c^{\mathsf{choice}} \notin \mathfrak{C}$, return $\perp$.
2. Store $c_i^{\mathsf{choice}} \leftarrow c^{\mathsf{choice}}$
3. $I \leftarrow I \cup \{i\}$
4. Return $\mathsf{success}$

On $(\mathsf{compute}, b)$ from S do:

1. If $b = 0$, return $\perp$.
2. Return res $\leftarrow f^{\mathrm{res}}(f^{\mathsf{agg}}((c_i^{\mathsf{choice}})_{i \in I}))$

Figure 3.1.: Ideal voting protocol.

bound on δ for all tallying-hiding voting protocols (where the results are of the form considered below), we now determine the optimal value of δ for the ideal (tally-hiding) voting protocol.

We have already presented the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\mathrm{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ in Section 3.1.1. We now formally analyze how the privacy level $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ of the ideal voting protocol depends on the specific result function f^{res} concerning the number of voters n_{voters}, the number of honest voters $n_{\mathsf{voters}}^{\mathsf{honest}}$, and the probability distribution μ according to which the honest voters select their choices. In the following, we will define the function $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$.

Recall that privacy is defined w.r.t. an honest voter, called the voter under observation, for which the adversary has to decide whether this voter voted for c_0^{choice} or c_1^{choice}, for any choices c_i^{choice} and c_j^{choice} from $\mathfrak{C}$.

Let $\mathsf{V}_{\mathsf{honest}}$ be the set of honest voters, which does not include the voter under observation. This set is of size $|\mathsf{V}_{\mathsf{honest}}| = n_{\mathsf{voters}}^{\mathsf{honest}}$. These honest voters will vote using the vote distribution μ. Furthermore, with $\mathsf{V}_{\mathsf{dishonest}}$, we denote the set of dishonest voters.

In the following, we will capture all possible choices of the voters from the choice space $\mathfrak{C}$ in a modified single-choice choice space $\mathfrak{C}_{\mathsf{Single}}(\mathfrak{C})$ with $|\mathfrak{C}|$ many choice components, where we interpret the first choice component as abstain. Using $\mathfrak{C}_{\mathsf{Single}}(\mathfrak{C})$ allows to notate the set of votes of the honest voters $(c_j^{\mathsf{choice}})_{j \in \mathsf{V}_{\mathsf{honest}}}$ as an aggregated tally $\vec{H} = (\vec{H}_i)_{i=0}^{|\mathfrak{C}|} = (c_{i,\mathsf{agg}}^{\mathsf{component}})_{i=0}^{|\mathfrak{C}|}$ over the equivalent votes in $\mathfrak{C}_{\mathsf{Single}}(\mathfrak{C})$, where $\vec{H}_i$ denotes the i-th entry of $\vec{H}$, $c_{0,\mathsf{agg}}^{\mathsf{component}}$ denotes the number of abstains, and for $i > 0$, $c_{i,\mathsf{agg}}^{\mathsf{component}}$ denotes the number of votes for the corresponding choice in $\mathfrak{C}$. In the same way, with $\vec{D}$, we denote the aggregated tally of the votes of the dishonest voters $(c_j^{\mathsf{choice}})_{j \in \mathsf{V}_{\mathsf{dishonest}}}$.

We are interested in the probability that, given the fixed choice c_i^{choice} of the voter under observation, and votes of the dishonest voters $\vec{D}$, the votes of the honest voters $\vec{H}$ lead in combination with the votes of the dishonest voters to the election result $\mathsf{elecres}$ under the election result function f^{res}. This probability depends on the vote distribution μ, and we denote this probability with $p_{\mathsf{elecres}}^{c_i^{\mathsf{choice}},\vec{D},\mu}$.

Essentially, $p_{\mathsf{elecres}}^{c_i^{\mathsf{choice}},\vec{D},\mu}$ denotes the probability, that the votes of the honest voters lead to a voting vector $\vec{H}$ such that $\vec{H}$, in combination with the votes of the dishonest voters $\vec{D}$ (denoted as $\vec{H} + \vec{D}$), results in $\mathsf{elecres}$. We denote the probability that the votes of the honest voters lead to an aggregated tally $\vec{H}$ over $\mathfrak{C}_{\mathsf{Single}}(\mathfrak{C})$ with $p_{\vec{H}}^{c_i^{\mathsf{choice}},\mu}$.

Now, we have

$$p_{\mathsf{elecres}}^{c_i^{\mathsf{choice}},\vec{D},\mu} = \sum_{\vec{H}\colon f^{\mathrm{res}}(\vec{H}+\vec{D})=\mathsf{elecres}} p_{\vec{H}}^{c_i^{\mathsf{choice}},\mu}$$

and

$$\begin{aligned} p_{\vec{H}}^{c_i^{\mathsf{choice}},\mu} &= \binom{n_{\mathsf{voters}}^{\mathsf{honest}}}{\vec{H}_0,\ldots,\vec{H}_{i-1},\vec{H}_i-1,\vec{H}_{i+1},\ldots,\vec{H}_{|\mathfrak{C}|}} \cdot p_{\mu,0}^{\vec{H}_0} \cdot \ldots \cdot p_{\mu,i}^{\vec{H}_{i-1}} \cdot p_{\mu,i}^{\vec{H}_i-1} \cdot p_{\mu,i}^{\vec{H}_{i+1}} \cdot \ldots \cdot p_{\mu,|\mathfrak{C}|}^{\vec{H}_{|\mathfrak{C}|}} \\ &= \frac{n_{\mathsf{voters}}^{\mathsf{honest}}!}{\vec{H}_1! \cdot \ldots \cdot \vec{H}_{|\mathfrak{C}|}!} \cdot p_{\mu,0}^{\vec{H}_0} \cdot \ldots \cdot p_{\mu,|\mathfrak{C}|}^{\vec{H}_{|\mathfrak{C}|}} \cdot \frac{\vec{H}_i}{p_{\mu,i}} \end{aligned}$$

Moreover, let $M_{c_i^{\mathsf{choice}},c_j^{\mathsf{choice}}}^{\vec{D},\mu} = \{\mathsf{elecres} \mid p_{\mathsf{elecres}}^{c_i^{\mathsf{choice}},\vec{D},\mu} \leq p_{\mathsf{elecres}}^{c_j^{\mathsf{choice}},\vec{D},\mu}\}$. Then, the intuition behind the definition of $\delta_{n_{\mathsf{voters}},n_{\mathsf{voters}}^{\mathsf{honest}},\mu}^{\mathrm{ideal}}(\mathfrak{C}, f^{\mathrm{res}})$ is as follows: If the observer, given an output $\mathsf{elecres}$, wants to decide whether the observed voter voted for choice c_i^{choice} or c_j^{choice}, the best strategy of the observer is to opt for c_j^{choice} if $\mathsf{elecres} \in M_{c_i^{\mathsf{choice}},c_j^{\mathsf{choice}}}^{\vec{D},\mu}$, i.e., the output is more likely if the voter voted for choice c_j^{choice}. We formalize this setting in the following definition:

$$\delta_{n_{\mathsf{voters}},n_{\mathsf{voters}}^{\mathsf{honest}},\mu}^{\mathrm{ideal}}(\mathfrak{C}, f^{\mathrm{res}}) := \max_{i,j \in \{1,\ldots,n_{\mathsf{choice}}\}} \max_{\vec{D}} \sum_{\mathsf{elecres} \in M_{c_i^{\mathsf{choice}},c_j^{\mathsf{choice}}}^{\vec{D},\mu}} \left(p_{\mathsf{elecres}}^{c_j^{\mathsf{choice}},\vec{D},\mu} - p_{\mathsf{elecres}}^{c_i^{\mathsf{choice}},\vec{D},\mu} \right)$$

Theorem 3.1. *The ideal protocol* $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\mathrm{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ *achieves a privacy level of* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$. *Moreover, it does not achieve* δ*-privacy for any* $\delta < \delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$.

Proof. See [KLM+20a]. □

Theorem 3.1 states this privacy level is ideal. More precisely, we show in [KLM+20a] that the ideal voting protocol achieves $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$-privacy and this privacy level is ideal, namely there exists no $\delta < \delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ such that the ideal protocol achieves δ-privacy.

In what follows, we give more examples to illustrate the effect of hiding the tally on the ideal privacy level. We prove that the derived formula is ideal in [KLM+20a].

3.1.3. Impact of Hiding the Tally

In this comparison, we assess the levels of privacy provided by the ideal protocol for three commonly used result functions computed in a tally-hiding manner:

1. The first function f_{AggTally} publishes the entire election result, including the number of votes per candidate. This function is commonly used in verifiable e-voting systems like Helios. We refer to the level of privacy with $\delta^{f_{\mathsf{AggTally}}}_{\mathrm{ideal}}$.

2. The second function is $f^{\mathrm{res}}_{\mathsf{Ranking}}$, which publishes the ranking of all candidates but not the number of votes per candidate. We refer to the level of privacy with $\delta^{f^{\mathrm{res}}_{\mathsf{Ranking}}}_{\mathrm{ideal}}$.

3. The third and final function $f^{\mathrm{res}}_{\mathsf{Plurality}}$ only publishes the election's winner without revealing the number of votes. We refer to the level of privacy with $\delta^{f^{\mathrm{res}}_{\mathsf{Plurality}}}_{\mathrm{ideal}}$.

In general, more information means less privacy. Depending on the distribution on the candidates, in general $\delta^{f_{\mathsf{AggTally}}}_{\mathrm{ideal}}$ is bigger than $\delta^{f^{\mathrm{res}}_{\mathsf{Ranking}}}_{\mathrm{ideal}}$ which in turn is bigger than $\delta^{f^{\mathrm{res}}_{\mathsf{Plurality}}}_{\mathrm{ideal}}$; see Figure 3.2 for an example.

Revealing the aggregated tally can lead to much worse privacy. Figure 3.2 already demonstrates that revealing the complete result leads to worse privacy. Figure 3.3 is another more extreme example. In both cases, honest voters favor one candidate.

The balancing attack. In some cases, there can be a massive difference between the privacy level of a voting system that uses tally-hiding functions and one that does not. This difference can be particularly significant if one choice has a higher probability. However, an adversary can use a balancing attack to cancel out the advantage of tally-hiding functions in terms of the privacy of single voters.

The balancing attack involves using dishonest voters to balance the probabilities for candidates. For example, consider an election with ten honest voters and two candidates, where the first candidate has a probability of 0.9. If an adversary instructs eight dishonest voters to vote for the second candidate, the expected total number of votes for each candidate is nine. In this case, the voter's choice under observation is crucial for the election's outcome.

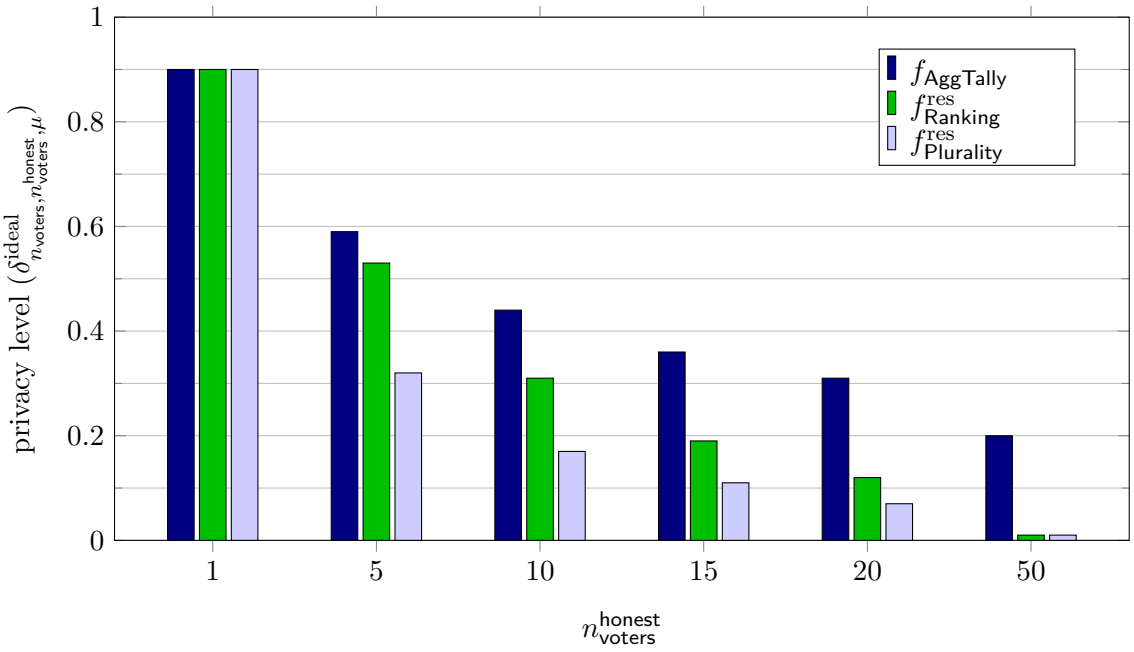


Figure 3.2.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for $n_{\text{cand}} = 3$, $\mathfrak{C}_{\text{Single}}$, $\mu(c_1^{\text{cand}}) = 0.6$, $\mu(c_2^{\text{cand}}) = 0.3$, $\mu(c_3^{\text{cand}}) = 0.1$, $\mu(\text{abstain}) = 0$, and $n^{\text{dishonest}}_{\text{voters}} = 0$.

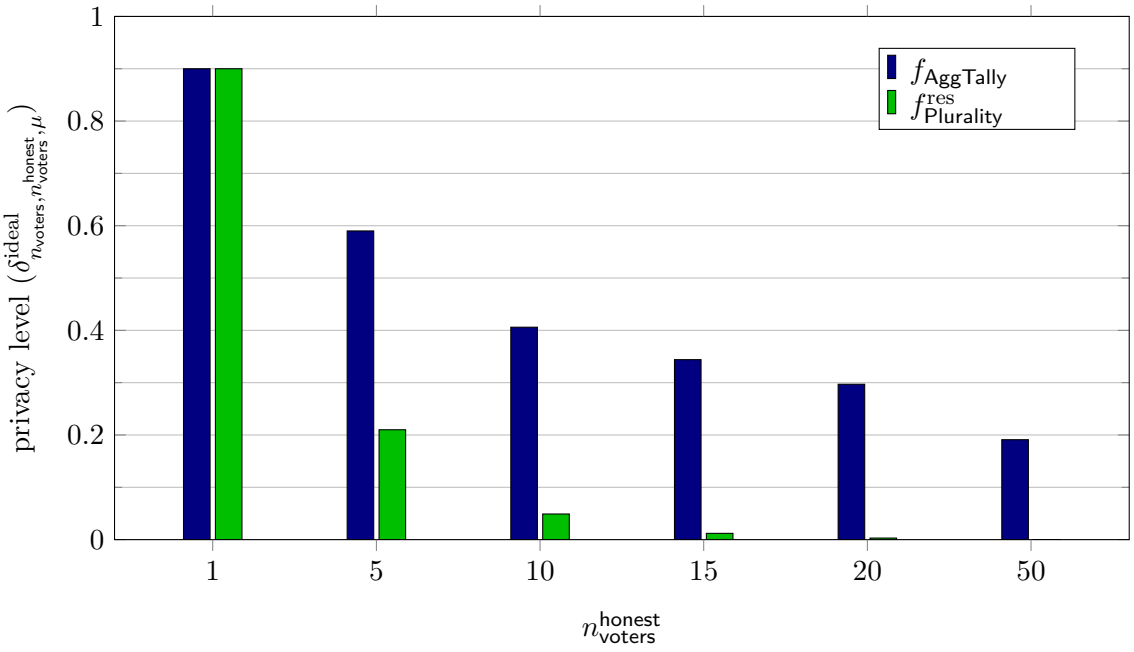


Figure 3.3.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for $n_{\text{cand}} = 2$, $\mathfrak{C}_{\text{Single}}$, $\mu(c_1^{\text{cand}}) = 0.1$, $\mu(c_2^{\text{cand}}) = 0.6$, $\mu(\text{abstain}) = 0.3$, and $n^{\text{dishonest}}_{\text{voters}} = 0$.

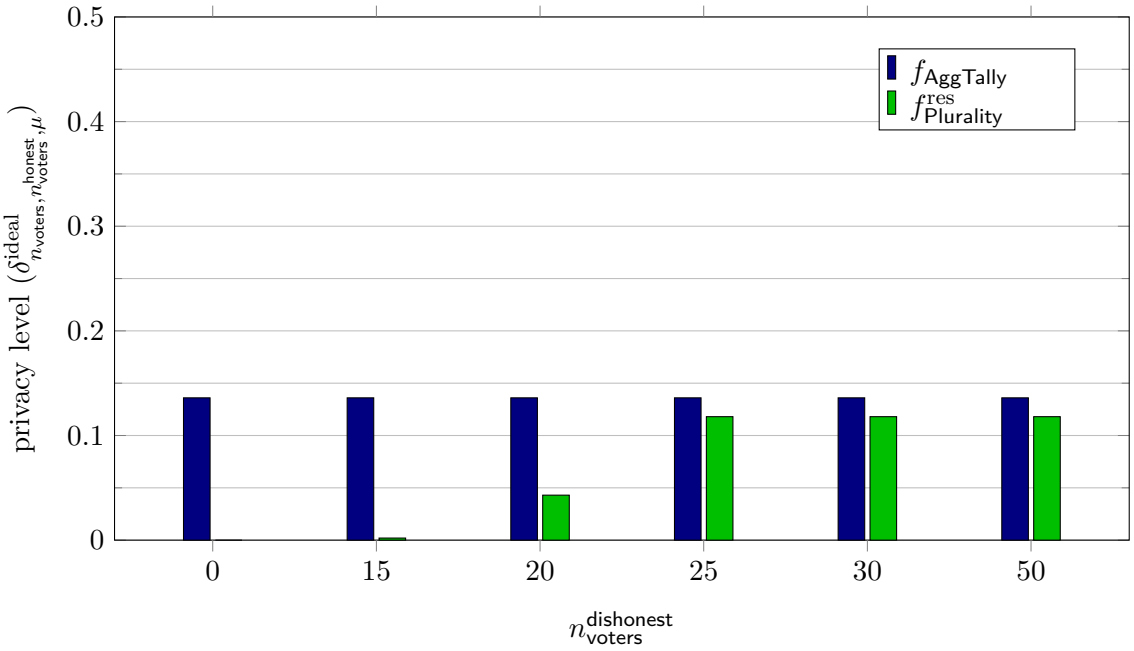


Figure 3.4.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for $n_{\text{cand}} = 2$, $\mathfrak{C}_{\text{Single}}$, $n^{\text{honest}}_{\text{voters}} = 100$, $\mu(c^{\text{cand}}_1) = 0.1$, $\mu(c^{\text{cand}}_2) = 0.6$, and $\mu(\text{abstain}) = 0.3$.

Although the number of dishonest voters is typically small compared to the number of honest voters, this balancing attack can be practical in small-scale elections with a few voters and candidates. Figure 3.4 illustrates an example of this.

Sometimes ranking is not better than the aggregated tally. If μ is a uniform distribution over the candidates, it is easy to show that $\delta^{f_{\text{AggTally}}}_{\text{ideal}} = \delta^{f^{\text{res}}_{\text{Ranking}}}_{\text{ideal}}$. The reason is that the best strategy for the adversary to decide whether the observed voter voted for c^{cand}_i or c^{cand}_j is to choose c^{cand}_i if c^{cand}_i gets more votes than c^{cand}_j, and this strategy is applicable even if only the ranking is published. We note that $f^{\text{res}}_{\text{Plurality}}$ is still better, i.e., $\delta^{f^{\text{res}}_{\text{Plurality}}}_{\text{ideal}} < \delta^{f_{\text{AggTally}}}_{\text{ideal}} = \delta^{f^{\text{res}}_{\text{Ranking}}}_{\text{ideal}}$. Figure 3.5 provides a concrete example.

In summary, as illustrated, even with only 15 honest voters, the level of privacy does not decrease much when the adversary changes the honest votes by only a few. Conversely, the result function can very well affect the level of privacy of a tally-hiding system: whether only the winner of an election is announced or the complete result typically significantly affects privacy.

Furthermore, we instantiate $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for complex ranked-choice voting methods. We present ideal privacy levels for several Condorcet methods in Figure 3.6.

Additionally, we instantiate $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for IRV. We depict concrete values of the ideal privacy level for the instant-runoff election function for two different distributions in Figure 3.7. While one distribution is uniform, we model the other more realistically: we sorted the candidates into a political spectrum. We assumed that if a voter chooses a candidate as her first preference, she will likely rank a candidate with a political opinion high in her ranking.

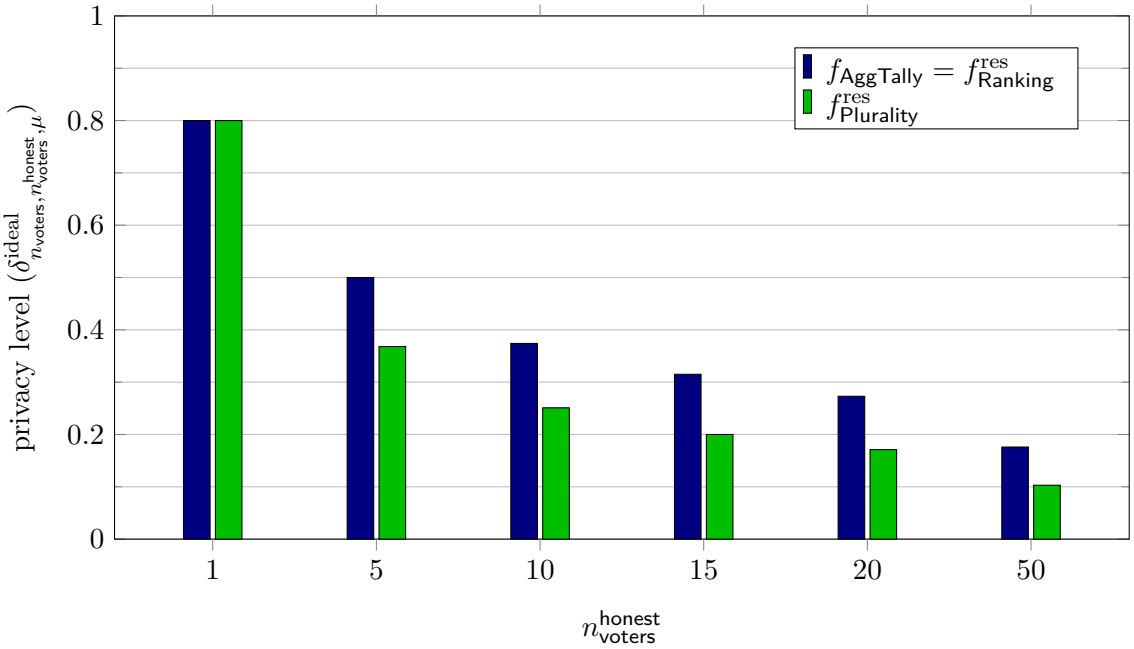


Figure 3.5.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for $n_{\text{cand}} = 5$, $\mathfrak{C}_{\text{Single}}$, and μ as a uniform distribution on the candidates with $\mu(\text{abstain}) = 0$, and $n^{\text{dishonest}}_{\text{voters}} = 0$.

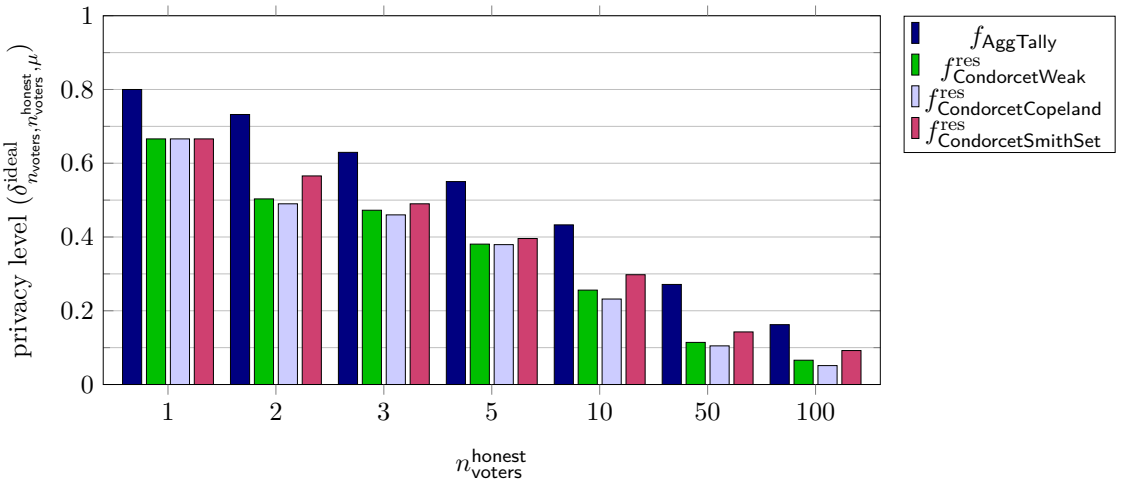


Figure 3.6.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for the ideal protocol with three candidates, uniform vote distribution, and no dishonest voters.

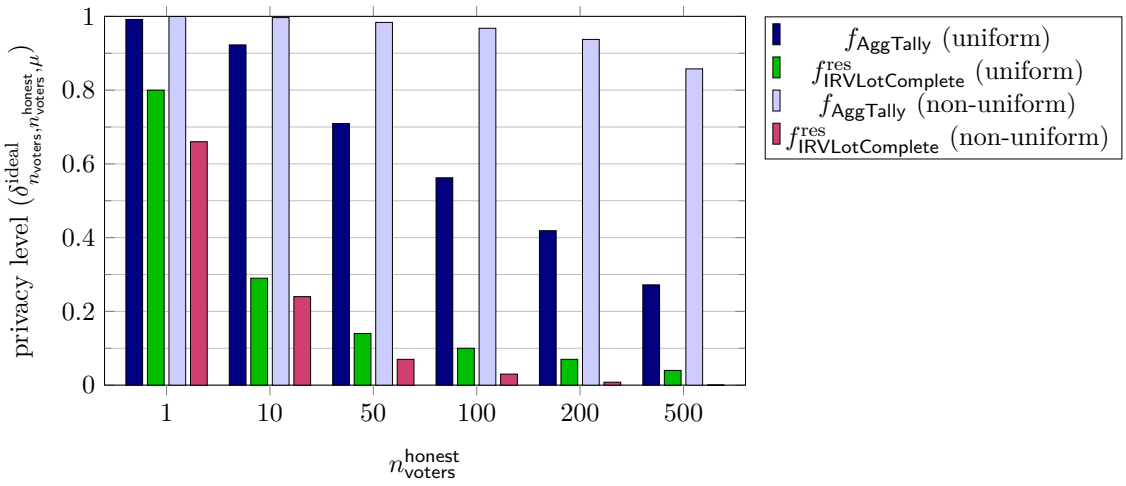


Figure 3.7.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for the ideal protocol with 5 candidates, IRV, and no dishonest voters. f_{AggTally} reveals the complete tally and $f^{\text{res}}_{\text{IRVLotComplete}}$ only reveals the winner of the election.

In detail, the non-uniform distribution in Figure 3.7 is as follows: the probabilities for the first rank are $(0.3, 0.2, 0.2, 0.12, 0.18)$. If the first rank is candidate 1, the probability that the second rank is candidate 2 is 0.95. Analogously for the pairs of first and second rank $(2,1), (4,5), (5,4)$. All remaining choices are taken uniformly at random. More precisely:

$$
\begin{aligned}
\mathsf{Pr}(1,2,*,*,*) &= 0.3 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(2,1,*,*,*) &= 0.2 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(1,3/4/5,*,*,*) &= 0.3 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(2,3/4/5,*,*,*) &= 0.2 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(3,*,*,*,*) &= 0.2 \cdot 4^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(4,1/2/3,*,*,*) &= 0.12 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(4,5,*,*,*) &= 0.12 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(5,1/2/3,*,*,*) &= 0.18 \cdot 0.05 \cdot 3^{-1} \cdot 3^{-1} \cdot 2^{-1} \\
\mathsf{Pr}(5,4,*,*,*) &= 0.18 \cdot 0.95 \cdot 3^{-1} \cdot 2^{-1}.
\end{aligned}
$$

As the figure demonstrates, disclosing the aggregated tally leads to a deficient level of privacy, yielding severe privacy issues, such as Italian attacks. Comparing these levels with the ones revealing only the winner confirms that voting systems that hide the tally for ranked-choice voting methods provide dramatically more favorable vote privacy than those that always publicize the aggregated tally. Even more, we witness that hiding the tally is, in fact *necessary* to achieve a proper privacy level for instant-runoff elections that include more than a few candidates.

We finally note that these results yield a lower bound for the privacy level of fully tally-hiding systems in general.

3.1.4. Formal Definition of Full Tally-Hiding

We now define this notion formally. We assume some set T of trustees and some threshold t.

Definition 3.1 (Full Tally-Hiding)**.** *Let* P *be a voting protocol with a set of trustees* T *and* $t \leq |\mathsf{T}|$*. We define that* P *is* full tally-hiding w.r.t. f^{res} and (T, t) *if (and only if), under the condition that at most* $t-1$ *parties in* T *are dishonest,* P *achieves* $\delta^{ideal}_{n_{\text{voters}}, n^{\text{honest}}_{\text{voters}}, \mu}(\mathfrak{C}, f^{\text{res}})$*-privacy.*

If a voting scheme achieves full tally-hiding, its privacy level is as good as the ideal voting protocol. Moreover, the election result function highly influences privacy, as demonstrated in Section 3.1.3.

3.2. Ordinos

This section presents the first provably secure, verifiable, and accountable fully tally-hiding e-voting system, called Ordinos. Conceptually, Ordinos is a framework following the general structure of the Helios remote e-voting system, at least in its first phase, but strictly extends Helios' functionality: Helios computes f_{AggTally}, that is, Helios always reveals the aggregated tally (see Section 2.7), no matter what the actual desired election result is. In contrast, Ordinos supports several election result functions in a fully tally-hiding fashion. That is, as explained in Section 3.1.4, beyond the result of the election according to the election result function f^{res}, Ordinos does not reveal any further information. In particular, Ordinos, unlike Helios, does not publish the aggregated tally (e.g., the number of votes per candidate) if not required by the result function.

Ordinos works as follows: Voters encrypt their votes and send their ciphertexts to a bulletin board. The trustees homomorphically aggregate the ciphertexts to obtain ciphertexts that encrypt the number of votes per candidate. Then, by an MPC protocol, the trustees evaluate the desired result function on these ciphertexts and publish the election result.

Compared to Helios, Ordinos uses different (instantiations of) cryptographic primitives and also additional primitives, in particular, a suitable MPC component, to obtain a tally-hiding system.

We carry out a detailed cryptographic analysis proving that Ordinos provides privacy, verifiability, and accountability: We show that Ordinos preserves the same level of verifiability/accountability as Helios under the same trust assumptions (namely that adversary does not corrupt the verification devices and the bulletin board), independently of the result function considered. More generally, this demonstrates that the standard definitions of verifiability and accountability can be achieved independently of whether we compute a result function in a tally-hiding way. Conversely, due to full tally-hiding, the level of privacy of Ordinos can be much better than for Helios, e.g., if only the winner or the ranking of candidates is to be published.

Our cryptographic analysis of Ordinos (for privacy and verifiability/accountability) is based on generic properties of the employed cryptographic primitives. Hence, it is general and does not rely on specific instantiations. To obtain a workable system, we carefully crafted instantiations using, among others, Paillier encryption [Pai99], an MPC protocol for greater-than tests by Lipmaa and Toft [LT13], as well as NIZKPs by Schoenmakers and Veeningen [SV15]. Based on this instantiation, we provide a proof-of-concept implementation of Ordinos and evaluate its performance, demonstrating its practicability, available at [LAA+23a].

We organize this section as follows. In Section 3.2.1, we present the Ordinos voting protocol on the conceptual level. In Section 3.2.2, we prove that Ordinos is verifiable, accountable, and full tally-hiding. In Section 3.2.3, we instantiate the Ordinos framework for various (real-world) elections. In Section 3.2.4, we modify the Ordinos framework to support voting in constituencies, and in Section 3.2.5, we present a web-based instantiation of Ordinos.

3.2.1. The Ordinos Framework

In this section, we present the Ordinos voting protocol on the conceptual level. Instead of relying on specific primitives, the security of Ordinos can be guaranteed under certain assumptions these primitives must satisfy. In particular, we can instantiate them with the most appropriate primitives available.

Ordinos extends the prominent Helios e-voting protocol. While Helios computes f_{AggTally}, that is, the complete election result is published (the number of votes per candidate/party), Ordinos supports tally-hiding elections. More specifically, the generic version of Ordinos, which we prove secure, supports arbitrary result functions evaluated over the aggregated votes per candidate/party and computes these result functions in a tally-hiding way. Our concrete instantiation then realizes many such practically relevant functions, see Section 3.2.3.

3.2.1.1. Protocol Participants

We run the Ordinos protocol among the following participants: a voting authority Auth, (human) voters $\mathsf{V}_1, \ldots, \mathsf{V}_{n_{\mathsf{voters}}}$, voter-supporting devices $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$, voter-verification devices $\mathsf{VVD}_1, \ldots, \mathsf{VVD}_{n_{\mathsf{voters}}}$, trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$, an authentication server AS, and an append-only bulletin board BB.

As further described below, the role of each (untrusted) voter-supporting device VSD is to generate and submit the voter's ballot, whereas the (trusted) voter-verification device VVD checks that the VSD behaved correctly. The role of the trustees is to tally the voters' ballots. In order to avoid a single trustee knowing how every single voter voted, the secret tallying key is distributed among all of them so that t out of n_{trustees} trustees need to collaborate to tally the ballots.

We assume the existence of the following authenticated channels:

- All parties have unilaterally authenticated channels to the bulletin board BB, ensuring that all parties have the same view on the bulletin board.
- For all VSD_i, an authenticated channel between this device and the authentication server AS, allowing AS to ensure that only eligible voters can cast their ballots.
- Each voter v_i has authenticated channels to the corresponding VSD_i as well as the VVD_i, modeling direct interaction.

By assuming such authenticated channels, we abstract away from the exact method the voters use to authenticate; in practice, several methods can be used, such as one-time codes, passwords, or external authentication services.

3.2.1.2. Protocol Overview

A protocol run consists of the following phases: In the *setup phase*, parameters and key shares are fixed and generated, and the public key shares are published. In the *voting phase*, voter v_i picks her choice $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C}$ accordingly to the choice space $\mathfrak{C}$, encrypts the choice $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}) = (E_{\mathsf{pk}}(c_{1,i}^{\mathsf{component}}), \ldots, E_{\mathsf{pk}}(c_{n_{\mathsf{components}},i}^{\mathsf{component}}))$, and then either audit or submit her ballot. Now or later, in the *voter verification phase*, voters verify that the authentication server has published their ballots. In the *tallying phase*, the trustees homomorphically aggregate the ballots to obtain $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_i = \bigoplus_{j=1}^{n_{\mathsf{voters}}} E_{\mathsf{pk}}(c_{i,j}^{\mathsf{component}})$ for $i \in \{1, \ldots, n_{\mathsf{components}}\}$, and then run a publicly accountable MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ on input $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_1, \ldots, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_{n_{\mathsf{components}}}$ in order to secretly compute, and publish the election result according to the result function f^{res} so that not even the trustees learn anything beyond the final result (which guarantees tally-hiding). Finally, in the *public verification phase*, everyone can verify that the trustees had tallied correctly.

We now explain each phase in more detail.

3.2.1.3. Setup Phase

In this phase, all election parameters are fixed and posted on the bulletin board by the voting authority Auth: the list $\vec{id}$ of eligible voters, opening and closing times, the election ID id_{election}, etc. as, well as the set $\mathfrak{C} \subseteq \mathbb{N}^{n_{\mathsf{components}}} \cup \{\mathsf{abstain}\}$ of valid choices where $n_{\mathsf{components}}$ denotes the number of (choice) components and $\mathsf{abstain}$ models that a voter abstains from voting. Furthermore, the $\mathsf{Ordinos}$ framework uses as a parameter a deterministic polynomial time function $f^{\mathrm{res}} : \{0,1\}^* \to \{0,1\}^*$ which computes the *final result* of the election based on a set of aggregated votes, i.e., the tally of the election. In Section 3.2.3 presents the election result functions that our instantiations of the $\mathsf{Ordinos}$ framework support.

The authentication server AS and each trustee T_k run the key generation algorithm of the digital signature scheme $\mathcal{S}$ to generate their public/private (verification/signing) keys. The verification keys are published on the bulletin board BB. In what follows, we implicitly assume

that whenever the authentication server AS or a trustee T_k publishes information, they sign this data with their signing keys.

Every trustee T_k runs the key share generation algorithm of the public-key encryption scheme $\mathcal{E}$ to generate its public/private (encryption/decryption) key share pair $(\mathsf{pk}_k, \mathsf{sk}_k)$. Additionally, each trustee T_k creates a NIZKP $\pi_k^{\mathsf{KeyShareGen}}$ to prove knowledge of sk_k and validity of $(\mathsf{pk}_k, \mathsf{sk}_k)$. Each trustee T_k then posts $(\mathsf{pk}_k, \pi_k^{\mathsf{KeyShareGen}})$ on the bulletin board BB. With $\mathsf{PublicKeyGen}$, everyone can compute the (overall) public key pk.

3.2.1.4. Voting Phase

In this phase, every voter v_i can decide to abstain from voting or to vote for some choice $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C} \subseteq \mathbb{N}^{n_{\mathsf{components}}}$. In the latter case, the voter inputs c_i^{choice} to her voter-supporting device VSD_i, which proceeds as follows. First, VSD_i encrypts each entry of c_i^{choice} separately under the public key pk and obtains a ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}) = (E_{\mathsf{pk}}(c_{1,i}^{\mathsf{component}}), \ldots, E_{\mathsf{pk}}(c_{n_{\mathsf{components}},i}^{\mathsf{component}}))$. That is, the j-th ciphertext in $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ encrypts the number of votes/points assigned by voter v_i to (choice) component j. After that, in addition to $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$, VSD_i creates a NIZKP $\pi_i^{\mathfrak{C}}$ in order to prove that it knows which choice c_i^{choice} the vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ encrypts and that $c_i^{\mathsf{choice}} \in \mathfrak{C}$. Finally, VSD_i sends a message to v_i to indicate that a ballot $\mathsf{b}_i = (id, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}), \pi_i^{\mathfrak{C}})$ is ready for submission, where $id \in \vec{id}$ is the voter's identifier. Upon receiving this message, the voter v_i can decide to either audit or submit the ballot b_i (*Benaloh challenge* [Ben07]), as described in what follows. We note that, beyond Benaloh challenges, several techniques exist in the literature (e.g., verification codes) to enable human voters to verify whether their ballots were cast by the voting devices as intended. Since these techniques are (typically) independent from whether the full tally is revealed or hidden, $\mathsf{Ordinos}$ could also be equipped with a different cast-as-intended mechanism than Benaloh challenges.

If v_i wants to audit b_i, v_i inputs an audit command to VSD_i who is supposed to reveal all the random coins that it had used to encrypt v_i's choice and to generate the NIZKPs. After that, v_i forwards this data and her choice c_i^{choice} to her verification device VVD_i which is supposed to check the correctness of the ballot, i.e., whether c_i^{choice} chosen by v_i and the revealed randomness by VSD_i yield b_i. The voter can not cast a ballot that she audited. Therefore, the voter is asked to vote again.

If v_i wants to submit b_i, v_i inputs a cast command to VSD_i, which is supposed to send b_i to the authentication server AS on an authenticated channel. If AS receives a ballot in the correct format (i.e., $id \in \vec{id}$ and id belongs to v_i, and the voter tagged b_i with the correct election ID id_{election}) and the NIZKP $\pi_i^{\mathfrak{C}}$ is valid, then AS responds with an acknowledgment consisting of a signature on the ballot b_i; otherwise, it does not output anything. Just as for Helios, variants of the protocol are conceivable where the voter's ID is not part of the ballot and not put on the bulletin board or at least not next to her ballot (see, e.g., [KTV12]). After that, VSD_i forwards the ballot b_i as well as the acknowledgment to VVD_i for verification purposes later on. If the

voter tried to re-vote and AS already sent out an acknowledgment, then AS returns the old acknowledgment only and does not accept the new vote. If VVD_i does not receive a correct acknowledgment from AS via VSD_i, it outputs a message to v_i who then tries to re-vote, and, if this does not succeed, files an authentication complaint on the bulletin board. If a party posts such a complaint, it is, in general, impossible to resolve the dispute and decide whom exactly to blame: (*i*) AS who might not have replied as expected (but claims, for instance, that the voter did not cast the ballot), or (*ii*) VSD_i who might not have submitted a ballot or forwarded the (correct) acknowledgment to VVD_i, or (*iii*) the voter who might not have cast a ballot but claims that she has. Note that this general problem applies to virtually any remote voting protocol. In practice, the voter could ask the voting authority Auth to resolve the problem.

When the voting phase is over, AS creates the list of submitted and valid ballots $\vec{\mathsf{b}}$. Then AS removes all ballots from $\vec{\mathsf{b}}$ that are duplicates w.r.t. the pair $(E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}), \pi_i^{\mathfrak{C}})$ only keeping the first one in order to protect against *replay attacks*, which jeopardize vote privacy [CS11]. Afterward, AS signs $\vec{\mathsf{b}}$ and publishes it on the bulletin board.

3.2.1.5. Voter Verification Phase

After the AS publishes the list of ballots $\vec{\mathsf{b}}$, each voter v_i can use her VVD_i to check whether (*i*) her ballot b_i appears in $\vec{\mathsf{b}}$ in the case she voted (if not, v_i can publish the acknowledgment she received from AS on the bulletin board which serves as binding evidence that AS misbehaved), or (*ii*) none of the ballots in $\vec{\mathsf{b}}$ contain her *id* in the case she abstained. In the latter case, the dispute cannot be resolved without further means: Did v_i vote but claim that she did not or did v_i not vote but AS used her *id* dishonestly? Variants of the protocol are conceivable where a voter is supposed to sign her ballot, and the authentication server presents such a signature in the case of a dispute (see, e.g., [CGGI14]). These methods also help in preventing so-called ballot stuffing.

In both cases, however, it is well-known that, realistically, not all voters are motivated enough to perform these verification procedures, and even if they are, they often fail to do so (see, e.g., [KOKV11]). In our security analysis of $\mathsf{Ordinos}$, we assume that voters perform the verification procedures with a certain probability. In order to increase verification rates, fully automated verification, as deployed in the sElect voting system [KMST16], turned out to be helpful and could be implemented in $\mathsf{Ordinos}$ as well.

3.2.1.6. Tallying Phase

The list of ballots $\vec{\mathsf{b}}$ posted by AS is the input to the tallying phase, which works as follows.

1. *Homomorphic Aggregation.* Each trustee T_k reads $\vec{\mathsf{b}}$ from the bulletin board BB and verifies its correctness (i.e., whether AS removed duplicates and invalid ballots). If this check fails, T_k aborts since AS should guarantee this. Otherwise, T_k homomorphically aggregates all vectors $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ in $\vec{\mathsf{b}}$ entrywise and obtains a ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}}) =$

$(E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_1, \ldots, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_{n_{\mathsf{components}}})$ with $n_{\mathsf{components}}$ entries each of which encrypts the number of votes/points of the respective (choice) component: $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_i = \bigoplus_{j=1}^{n_{\mathsf{voters}}} E_{\mathsf{pk}}(c_{i,j}^{\mathsf{component}})$ for $i \in \{1, \ldots, n_{\mathsf{components}}\}$.

2. *Secure Function Evaluation.* The trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$ run the publicly accountable MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ with input $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ to securely evaluate the result function f^{res}. They then output the election result according to f^{res} and a NIZKP of correct evaluation π^{MPC}. This NIZKP π^{MPC} typically consists of several NIZKPs, e.g., NIZKPs of correct decryption.

In our instantiation, we realized tallying functions f^{res} that we build on top of a secure *greater-than* MPC protocol.

3.2.1.7. Public Verification Phase

In this phase, every participant, including the voters or external observers, can verify the correctness of the tallying procedure, particularly the correctness of all NIZKPs.

3.2.2. Security of Ordinos

Many of the components that an Ordinos instantiation uses are not fixed by the Ordinos framework because they strongly depend on the specific election to apply. Specifically, the following parameters and components have to be specified or constructed by a protocol designer to create an instantiation of Ordinos for a concrete election: (i) the choice space $\mathfrak{C}$ and election result function f^{res}, (ii) a threshold encryption scheme $\mathcal{E}$, (iii) NIZKPs $\pi^{\mathsf{KeyShareGen}}$ and $\pi^{\mathfrak{C}}$, (iv) a EUF-CMA-secure signature scheme $\mathcal{S}$, and (v) a publicly accountable MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ for computing the election result function f^{res}.

In this section, we show that if the above component defined by protocol designers meets specific properties, the resulting Ordinos instance achieves verifiability, accountability, privacy, and full tally-hiding. We start by formally defining the Ordinos voting protocol.

3.2.2.1. Formal Protocol Model of Ordinos

In this section, we precisely define the honest programs of all agents in Ordinos.

Set of agents in Ordinos. The set of agents of $\mathsf{P}_{\mathsf{Ordinos}}$ consists of all agents described in Section 3.2.1, i.e., the bulletin board BB, n_{voters} (human) voters $\mathsf{v}_1, \ldots, \mathsf{v}_{n_{\mathsf{voters}}}$, voter supporting devices $\mathsf{VSD}_1, \ldots, \mathsf{VSD}_{n_{\mathsf{voters}}}$, voter verification devices $\mathsf{VVD}_1, \ldots, \mathsf{VVD}_{n_{\mathsf{voters}}}$, the authentication server AS, n_{trustees} trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$, and in addition, a scheduler S. The latter party plays the role of the voting authority Auth and schedules all other agents in a run according to the protocol phases. Also, it is the master program of $\mathsf{P}_{\mathsf{Ordinos}}$. Channels connect all agents with all other agents; honest agents will not use all these channels, but dishonest agents might. The honest programs $\mathfrak{p}_{\mathsf{a}}$ of honest agents a are defined in the way according to the description

of the agents in Section 3.2.1. We assume that the scheduler S and the bulletin board BB are honest. All other agents can be dishonest. These agents can run arbitrary probabilistic (polynomial-time) programs. We note that the scheduler is only a modeling tool. It does not exist in real systems. The assumption that the bulletin board is honest is common; Helios makes this assumption, too. In reality, the bulletin board should be implemented in a distributed way (see, e.g., [CS14, KKL+18]).

Scheduler S. In the protocol $\mathsf{P}_{\mathsf{Ordinos}}$, the honest program $\mathfrak{p}_{\mathsf{S}}$ of S plays the role of the master program. We assume that the scheduler knows which agents are honest and which are dishonest to schedule the agents appropriately. In what follows, we implicitly assume that the scheduler triggers the adversary (any dishonest party) at the beginning and end of the protocol run. Also, the adversary is triggered each time an honest party finishes its computations (after being triggered by the scheduler in some protocol step). Therefore, the adversary is up to date and can output its decision at the end of the run. By this, we obtain stronger security guarantees.

Similarly, we assume that the scheduler triggers the judge each time any other party (honest or dishonest) finishes its computation (after being triggered by the scheduler). Therefore, the judge can give its verdict after each protocol step. Suppose the judge posts a message on the bulletin board BB, which indicates to stop the whole protocol. In that case, the scheduler triggers the adversary once more (to allow it to output its decision) and then halts the entire system. We also let the scheduler create common reference strings (CRSs) for all the required NIZKPs. We call the setup algorithm of the non-interactive zero-knowledge proof systems used in the protocol and provide them to all parties.

In the remaining part of the section, we precisely describe the honest program of the scheduler depending on the voting phase.

Scheduling the setup phase. At the beginning of the election, the scheduler determines the set of possible choices defined as $\mathfrak{C} \subseteq \mathbb{N}^{n_{\mathsf{components}}} \cup \{\mathsf{abstain}\}$ of valid choices where $\mathsf{abstain}$ models that a voter abstains from voting. Then, the scheduler generates a random number id_{election}, the election identifier, with the length of the security parameter η and sends it to the bulletin board BB, which publishes id_{election} and $\mathfrak{C}$. Whenever a party computes a signature on some message m, this implicitly means that we calculate a signature on the tuple $(id_{\text{election}}, \mathsf{tag}, \mathsf{m})$ where id_{election} is an election identifier (different for distinct elections) and tag is a tag distinguishable for signatures with dissimilar purposes (for example, a signature on a list of voters uses a different tag than a signature on a list of ballots).

After that, the scheduler first triggers all honest trustees T_k, which are supposed to generate their verification/signing key pairs $(\mathsf{verify}_k, \mathsf{sign}_k)$ and publish the public verification keys verify_k on the bulletin board BB, and then all the dishonest ones. In what follows, we implicitly assume that each trustee T_k is supposed to sign all its messages to the bulletin board under sign_k.

Next, the scheduler triggers all honest trustees again, and then all dishonest ones, to run the key share generation algorithm $\mathsf{KeyShareGen}$ of the public-key encryption scheme $\mathcal{E}$. As a result, each trustee publishes a public key share pk_k (together with a NIZKP of correctness and

knowledge of the respective secret key share sk_k) so that we can obtain the public key pk by running $\mathsf{PublicKeyGen}$ on the published public key shares.

Scheduling the voting phase. The scheduler triggers all the honest voters and then the dishonest ones, allowing them to cast their ballots to the authentication server AS. After each step (when the voters and the authentication server finish their computations), the scheduler triggers the voter again to allow the voter to post a complaint if she does not get a valid acknowledgment from the authentication server. As specified below, we model the authentication server AS to provide all collected ballots (even before AS publishes them on the bulletin board BB) to an arbitrary participant who requests them. Afterward, the scheduler triggers the authentication server, which is supposed to publish the list of ballots $\vec{\mathsf{b}}$ (containing the (first) valid ballot cast by each eligible voter) on the bulletin board BB.

Scheduling the voter verification phase. Similarly to the voting phase, the scheduler triggers first the honest voters who are supposed to verify (with probability p_{verify}) the input to the tallying phase. See below for details. Afterward, the scheduler triggers all the dishonest voters.

Scheduling the tallying phase. The scheduler runs the scheduling procedure of the given MPC protocol.

Authentication Server AS*.* The authentication server AS, when triggered by the scheduler S in the key generation phase for the signature scheme, runs the key generation algorithm KeyGen of $\mathcal{S}$ to obtain a verification/signing key pair $(\mathsf{verify}_{\mathsf{AS}}, \mathsf{sign}_{\mathsf{AS}})$. Then, the authentication server sends the verification key to the bulletin board BB.

When the authentication server AS receives a ballot b_i from an eligible voter v_i via an authenticated channel, the server checks whether (i) the voter tagged the received ballot with the correct election identifier, (ii) the voter *id* belongs to the authenticated voter and has not been used before, (iii) the ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$ has not been submitted before, (iv) the NIZKPs are correct. If this holds, the acknowledgment consists of a signature under $\mathsf{sign}_{\mathsf{AS}}$ on the ballot b_i; otherwise, it does not output anything. The authentication server adds b_i together with the voter *id* to the (initially empty) list of ballots $\vec{\mathsf{b}}$. If a voter tried to re-vote and AS already sent out an acknowledgment, then AS returns the old acknowledgment only and does not consider the new vote.

When the scheduler S triggers the authentication server at the end of the voting phase, AS signs the list $\vec{\mathsf{b}}$ with $\mathsf{sign}_{\mathsf{AS}}$, and sends it, together with the signature, to the bulletin board.

Furthermore, to model the assumption that the channel from the voter to AS is authenticated but not (necessarily) secret, the authentication server AS is also supposed to provide all ballots collected so far to any requesting agent (even before AS published them on the bulletin board BB).

Bulletin Board BB*.* Running its honest program, the bulletin board BB accepts messages from all agents. If the bulletin board BB receives a message via an authenticated channel, it stores the message in a list along with the identifier of the agent who posted the message. Otherwise, if

the message is sent anonymously, it only stores the message. The bulletin board sends its stored content to the requesting agent upon request.

Voter v_i. A voter v_i, when triggered by the scheduler S in the voting phase, picks c_i^{choice} from $\mathfrak{C}$ according to the probability distribution μ. A choice may be a distinct value $\mathsf{abstain}$, which expresses abstention from voting, or an integer vector from $\mathbb{N}^{n_{\mathsf{components}}}$. The voter program stops if $c_i^{\mathsf{choice}} = \mathsf{abstain}$. Otherwise, if $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C}$, the voter enters c_i^{choice} to her voter-supporting device VSD_i. The voter expects a message from VSD_i indicating that a ballot b_i is ready for submission. After that, the voter decides (with a certain probability p_{audit}) whether she wants to audit or submit the b_i.

If v_i decides to submit b_i, she enters a message to her VSD indicating submission. The voter expects an acknowledgment from the authentication server AS via VSD_i. After that, the voter enters the acknowledgment to her verification device VVD_i, which checks its correctness. If the voter did not obtain an acknowledgment or VVD_i reports that the acknowledgment is invalid, the voter posts a complaint on the bulletin board via her authenticated channel. Note that the program of the voter may not get any response from VSD_i in case AS or VSD_i are dishonest. To enable the voter, in this case, to post a complaint on the bulletin board, the scheduler triggers the voter again (still in the voting phase).

If v_i decides to audit b_i, she enters a message to her VSD indicating auditing. The voter expects a list of random coins from VSD_i. After that, the voter enters her choice c_i^{choice}, the ballot b_i, and the list of random coins to her verification device VVD_i, which checks its correctness. If VVD_i returns that verification was unsuccessful, the voter posts a complaint on the bulletin board via her authenticated channel. In any case, the voter program returns to the start of the voting phase.

The voter v_i, when triggered by the scheduler S in the verification phase, carries out the following steps with probability p_{verify}. If c_i^{choice} was $\mathsf{abstain}$, the voter verifies that her id is not listed in the list of ballots $\vec{\mathsf{b}}$ output by the authentication server. She will file a complaint if this is not the case. If $c_i^{\mathsf{choice}} \neq \mathsf{abstain}$, the voter checks that her id and her ballot b_i appear in the list of ballots $\vec{\mathsf{b}}$, output by the authentication server. As before, she files a complaint if this is not the case.

Voter-supporting device VSD_i. If the voter supporting device obtains the vote $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C}$ from v_i, then VSD_i encrypts each integer $c_{j,i}^{\mathsf{component}}$ under the public key pk to obtain a ciphertext vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}})$. Afterward, the voter creates a NIZKP $\pi_i^{\mathfrak{C}}$ of knowledge and correctness for the $n_{\mathsf{components}}$-ary relation over the plaintext space which holds if and only if $(c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C} \setminus \{\mathsf{abstain}\}$. The voter-supporting device VSD_i stores all the random coins used to encrypt the vote and create the NIZKP. After that, VSD_i creates the ballot

$$\mathsf{b}_i = (id, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_i^{\mathsf{choice}}), \pi_i^{\mathfrak{C}}),$$

and returns a message to v_i indicating that her ballot b_i is ready for submission.

The VSD expects a message from the voter indicating submission or auditing. If the voter wants to submit b_i, then VSD_i sends b_i to the authentication server AS. VSD_i expects to get back an acknowledgment (a signature of AS on the submitted ballot) which it returns to v_i. If the voter wants to audit b_i, then VSD_i returns all the random coins that it used to create b_i and removes them from its internal storage afterward.

Voter-verification device VVD_i. If the voter verification device VVD_i gets as input a choice c_i^{choice}, a ballot b_i, and a list of random coins, then VVD_i verifies whether c_i^{choice} together with these random coins yield b_i. After that, VVD_i returns the result of this check.

Suppose the voter-verification device VVD_i gets as input an acknowledgment and a ballot b_i from v_i. In that case, it checks whether the acknowledgment is valid for b_i (i.e., that the acknowledgment is a valid signature by AS for b_i). After that, VVD_i returns the result of this check.

Trustee T_k. A trustee T_k, when triggered by the scheduler S in the key generation phase for the signature scheme, runs the key generation algorithm KeyGen of $\mathcal{S}$ to obtain a verification/signing key pair $(\mathsf{verify}_k, \mathsf{sign}_k)$. Then, the trustee sends the verification key to the bulletin board BB.

When triggered by the scheduler S in the key generation phase for the encryption scheme, the trustee T_k runs the key share generation algorithm $\mathsf{KeyShareGen}$ of $\mathcal{E}$ to obtain a secret key share sk_k and a public key share pk_k. Then, the trustee T_k creates a NIZKP $\pi_k^{\mathsf{KeyShareGen}}$ for proving correctness of the public key share pk_k including knowledge of an adequate secret key share sk_k. The trustee signs $(\mathsf{pk}_k, \pi_k^{\mathsf{KeyShareGen}})$ with the signing key sign_k and sends it, together with signature, to the bulletin board BB.

When triggered by the scheduler S in the tallying phase, the trustee T_k reads the list of ballots $\vec{\mathsf{b}}$ published and signed by the authentication server AS from the bulletin board BB. If no such list exists, the signature is incorrect, or if the list is incorrect (see above), the trustee aborts. Otherwise, T_k calculates

$$E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}}) = (E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_1, \ldots, E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_{n_{\mathsf{components}}})$$

where

$$E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_i = \bigoplus_{j=1}^{n_{\mathsf{voters}}} E_{\mathsf{pk}}(c_{i,j}^{\mathsf{component}})$$

Up to this step, Ordinos is identical to Helios.

Then, the trustees run the MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ with the input $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$. The output of $\mathsf{P}^{f^{\mathrm{res}}}$ is the overall election result of Ordinos, plus some NIZKP of correct evaluation π^{MPC}.

Judge J. We assume that J is honest. We note that the honest program $\mathfrak{p}_{\mathsf{J}}$ of J, as defined below, uses only publicly available information, and therefore, every party, including the voters and external observers, can run the judging procedure.

The program $\mathfrak{p}_{\mathsf{J}}$, whenever triggered by the scheduler S, reads data from the bulletin board and verifies its correctness, including the correctness of posted complaints. The judge outputs verdicts (as described below) on a distinct tape. More precisely, the judge outputs a verdict in the following situations:

(J1) If a party a deviates from the protocol specification in an obvious way, then J blames a individually by outputting the verdict $\mathsf{dis}(\mathsf{a})$. This is the case if the party a, for example, (*i*) does not publish data when expected, or (*ii*) publishes data that is not in the expected format, or (*iii*) publishes a NIZKP which is not correct.

(J2) If a voter v_i posts an authenticated complaint in the voting phase that she has not received a valid acknowledgment from the authentication server AS, then the judge outputs the verdict $\mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{VSD}_i) \vee \mathsf{dis}(\mathsf{AS})$, which means that (the judge believes that) one of the parties v_i, VSD_i, AS is dishonest but cannot determine which of them.

(J3) If a voter v_i posts an authenticated complaint claiming that she did not vote, but her name was posted by the authentication server AS in one of the ballots in $\vec{\mathsf{b}}$, the judge outputs the verdict $\mathsf{dis}(\mathsf{AS}) \vee \mathsf{dis}(\mathsf{v}_i)$.

(J4) If, in the verification phase, a valid complaint is posted containing an acknowledgment of AS, i.e., the complaint contains a signature of AS on a ballot that is not in the list of ballots $\vec{\mathsf{b}}$ published by AS, then the judge blames AS individually by outputting the verdict $\mathsf{dis}(\mathsf{AS})$.

(J5) During the execution of $\mathsf{P}^{f^{\text{res}}}$ the judge runs the judging procedure $\mathsf{J}_{\mathsf{MPC}}$ of $\mathsf{P}^{f^{\text{res}}}$. If $\mathsf{J}_{\mathsf{MPC}}$ outputs a verdict, then J also outputs this verdict.

(J6) If, in the submission phase, a voter v_i posts an authenticated complaint claiming that her VSD_i did not produce a correct ballot b_i for her chosen input, then the judge outputs the verdict $\mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{VSD}_i)$, which means that (the judge believes that) either v_i or VSD_i is dishonest but cannot determine which of them.

If none of these situations occur, the judge J outputs `accept` on a distinct tape.

Based on this formal model of the Ordinos voting framework, we now analyze the accountability and privacy level of Ordinos.

3.2.2.2. Accountability of Ordinos

Now, we can precisely analyze the accountability level (see Section 2.4) of Ordinos. For this, we first define the accountability constraints and property of Ordinos. Then, we state and prove the accountability theorem.

Accountability constraints. For Ordinos, we use the following accountability constraints:

- Let χ_i denote the set of runs of an instance of $\mathsf{P}_{\mathsf{Ordinos}}$ where voter v_i complains that she did not get a receipt from AS via VSD_i. In such runs, the judge cannot be sure who to blame individually:
  - AS who might not have replied as expected (but claims, for instance, that the voter did not cast the ballot) or
  - VSD_i who might not have submitted a ballot or forwarded the (correct) acknowledgment to VVD_i or
  - the voter who might not have cast a ballot but claims that she has. We note that this is a general problem that applies to virtually any remote voting protocol. In practice, the voter could ask the voting authority Auth to resolve the problem. This is captured by the accountability constraint $\chi_i \Rightarrow \mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{VSD}_i) \vee \mathsf{dis}(\mathsf{AS})$. Recall that we define that the judge J ensures this constraint in a run $\mathfrak{r}$ if $\mathfrak{r} \notin \chi_i$ or the verdict output by the J in r implies $\mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{VSD}_i) \vee \mathsf{dis}(\mathsf{AS})$ in the sense of propositional logic.
- Let χ_i' contain all runs of $\mathsf{P}_{\mathsf{Ordinos}}$ where the voter v_i complains that she did not vote, a ballot in $\vec{\mathsf{b}}$ published by AS contains her name. Then, the accountability constraint for this situation is $\chi_i' \Rightarrow \mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{AS})$.
- Let χ_i'' contain all runs of $\mathsf{P}_{\mathsf{Ordinos}}$ where the voter v_i complains that the ballot auditing was unsuccessful. Then, the accountability constraint for this situation is $\chi_i'' \Rightarrow \mathsf{dis}(\mathsf{v}_i) \vee \mathsf{dis}(\mathsf{VSD}_i)$.

The accountability theorem for $\mathsf{Ordinos}$ (see below) states that if the adversary breaks the goal $\gamma(k, \varphi)$ (as defined in Section 2.3) in a run of $\mathsf{P}_{\mathsf{Ordinos}}$ but neither χ_i, χ_i' nor χ_i'' occur (for some voter v_i), then (at least) one misbehaving party can be blamed individually (with a certain probability). The accountability constraint for this situation is

$$\neg\gamma(k, \varphi) \wedge \neg\chi \Rightarrow \mathsf{dis}(\mathsf{AS}) \mid \mathsf{dis}(\mathsf{T}_1) \mid \ldots \mid \mathsf{dis}(\mathsf{T}_{n_{\mathsf{trustees}}}),$$

where $\chi = \bigcup_{i \in \{1,\ldots,n_{\mathsf{voters}}\}} (\chi_i \cup \chi_i' \cup \chi_i'')$. Now, the judge J ensures this constraint in a run $\mathfrak{r}$ if $\mathfrak{r} \notin \neg\gamma(k, \varphi) \wedge \neg\chi$ or the verdict output by J in $\mathfrak{r}$ implies $\mathsf{dis}(\mathsf{a})$ for some party a mentioned in the constraint.

Accountability property. For $\mathsf{P}_{\mathsf{Ordinos}}$ and the goal $\gamma(k, \varphi)$, we define the accountability property $\Phi_k^{\mathsf{Ordinos}}$ to consist of the constraints mentioned above for the cases χ_i, χ_i', χ_i'' (for all $i \in \{1, \ldots, n_{\mathsf{voters}}\}$), and $\neg\gamma(k, \varphi) \wedge \neg\chi$. This accountability property covers $\neg\gamma(k, \varphi)$ by construction, i.e., if $\gamma(k, \varphi)$ is not satisfied, these constraints require the judge J to blame some party. Note that all verdicts are atomic in the runs covered by the last constraint of $\Phi_k^{\mathsf{Ordinos}}$. Thus, $\Phi_k^{\mathsf{Ordinos}}$ requires that except for the cases where χ occurs, whenever a run violates the goal $\gamma(k, \varphi)$, an individual party is blamed, so-called *individual accountability*. The NIZKPs and signatures used in $\mathsf{Ordinos}$ allow us to achieve individual accountability.

For the accountability theorem, we make the following assumptions:

(A1) The public-key encryption scheme $\mathcal{E}$ is correct (for verifiability, IND-CPA-security is not needed), $\pi^{\mathsf{KeyShareGen}}$ and π^{E}_{pk} are NIZKPs, and the signature scheme $\mathcal{S}$ is EUF-CMA-secure.

(A2) The scheduler S, the bulletin board BB, the judge J, and all voter-verification devices VVD_i are honest, i.e., $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{BB}) \wedge \bigwedge_{i=1}^{n_{\mathsf{voters}}} \mathsf{hon}(\mathsf{VVD}_i)$.

(A3) The MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ enjoys *individual accountability* (w.r.t. the goal $\gamma(0,\varphi)$ and accountability level 0), meaning that if the outcome of the protocol does not correspond to f^{res}, then the judge always can blame at least one of the trustees individually, because in this case the NIZKP π^{MPC} fails. Our instantiations presented in Section 3.2.3 fulfill this assumption.

Now, the following theorem states the accountability result of Ordinos.

Theorem 3.2 (Accountability of Ordinos)**.** *Under the assumptions (A1) to (A3) and the judging procedure run by the judge* J *stated above,* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, \mathfrak{C}, f^{\mathrm{res}})$ *is* $(\Phi_k^{\mathsf{Ordinos}}, \delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}))$*-accountable w.r.t. the judge* J *where*

$$\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}) = \max\left(1 - p_{\mathsf{verify}}, 1 - p_{\mathsf{audit}}\right)^{\lceil \frac{k+1}{2} \rceil}.$$

Proof. See [Mül19]. □

Verifiability follows directly from the fact that accountability implies verifiability. However, we can guarantee verifiability under weaker assumptions. That is, the MPC protocol must only be verifiable (instead of accountable), which we capture in the following modification of (A3) from above:

(V3) The MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ is $(\gamma(0,\varphi), 0)$-verifiable, meaning that if the output of $\mathsf{P}^{f^{\mathrm{res}}}$ does not correspond to its input, then we can always detect this publicly.

Theorem 3.3 (Verifiability of Ordinos)**.** *Under the assumptions (A1), (A2), (V3) and the judging procedure run by the judge* J *stated above,* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, \mathfrak{C}, f^{\mathrm{res}})$ *is* $(\gamma(k,\varphi), \delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}))$*-verifiable w.r.t. the judge* J *where*

$$\delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}) = \max\left(1 - p_{\mathsf{verify}}, 1 - p_{\mathsf{audit}}\right)^{\lceil \frac{k+1}{2} \rceil}.$$

Proof. Directly follows the proof of Theorem 3.2. □

3.2.2.3. Privacy and Tally-Hiding of Ordinos

We now prove that Ordinos provides a high level of privacy w.r.t. k-risk-avoiding adversaries and in the case that at most $t-1$ trustees are dishonest, where t is the decryption threshold of the underlying encryption scheme. If t trustees were dishonest, privacy cannot be guaranteed

because an adversary could decrypt every ciphertext in the list of ballots. By *high level of privacy*, we denote that Ordinos provides δ-privacy for a δ close to the ideal one.

More specifically, we formulate the formal privacy result for Ordinos w.r.t. the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\mathrm{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$, presented in Figure 3.1. Recall that in this protocol, honest voters pick their choices according to the distribution μ, and in every run, there are $n_{\mathsf{voters}}^{\mathsf{honest}}$ many honest voters and n_{voters} voters overall. The ideal protocol collects the votes of the honest voters and the dishonest ones (where the latter ones are independent of the votes of the honest voters) and outputs the result according to the result function f^{res}. In Section 3.1.2, we analyze the privacy level $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$. This ideal protocol has to depend on the given parameters.

Assumptions. In our analysis, we assume that honest voters do not abstain from voting to have a guaranteed number of votes by honest voters. Note that the adversary would know which voters abstained and which did not. To prove that the privacy level of Ordinos is essentially the ideal one, we make the following assumptions about the primitives we use:

(P1) The public-key encryption scheme $\mathcal{E}$ is IND-CPA-secure, the signatures are EUF-CMA-secure, and $\pi^{\mathsf{KeyShareGen}}$ and $\pi^{\mathfrak{C}}$ are NIZKPs.

(P2) The MPC protocol $\mathsf{P}^{f^{\mathrm{res}}}$ realizes (in the sense of universal composability [Can01, Küs06]) the ideal MPC protocol presented in Figure 3.8 which essentially takes as input a vector of ciphertexts and returns $f^{\mathrm{res}} \circ f^{\mathsf{agg}}$ evaluated on the corresponding plaintexts. The level of privacy of Ordinos depends on the number of ballots cast by honest voters.

(P3) The probability of abstention is 0 in μ.

(P4) The process $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}}$ of $\mathsf{P}_{\mathsf{Ordinos}}$, the set $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}})$ of admissible adversaries for $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}}$ is defined as follows. An adversary A belongs to $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}})$ if (and only if) it satisfies the following conditions:

- $\mathfrak{p}_{\mathsf{A}}$ is k-risk-avoiding for $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}}$,
- the probability that $\mathfrak{p}_{\mathsf{A}}$ corrupts more than $t-1$ trustees in a run of $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}} \| \mathfrak{p}_{\mathsf{A}}$ is negligible,
- the probability that $\mathfrak{p}_{\mathsf{A}}$ corrupts more than $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters in a run of $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}} \| \mathfrak{p}_{\mathsf{A}}$ is negligible, and
- the probability that $\mathfrak{p}_{\mathsf{A}}$ corrupts an honest voter's supporting or verification device is negligible.

We note that, in principle, the following privacy result also holds if the honest voters' verification devices are dishonest because, in our model, an honest voter chooses a *fresh* candidate according to μ each time after auditing her ballot. If, however, an honest voter always used her actual candidate when auditing her ballot, then the verification device needs to be honest for vote privacy, too.

$\underline{\mathcal{I}_{\mathsf{MPC}}(\mathcal{E}, f^{\mathsf{MPC}})}$

Parameters:

- A $(t, n_{\mathsf{trustees}})$-threshold public-key encryption scheme $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, E_{\mathsf{pk}}, \mathsf{dec}^{\mathsf{share}}, \mathsf{dec})$
- Choice Space $\mathfrak{C}$
- Function $f^{\mathsf{MPC}}\colon \{0,1\}^* \to \{0,1\}^*$
- Set of trustees $\mathsf{T} = \{\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}\}$
- Bulletin Board BB

KeyGen: On input `KeyGen` by each $\mathsf{T}_k \in \mathsf{T}$:

1. $\forall \mathsf{T}_k \in \mathsf{T}$:
   a) $(\mathsf{pk}_k, \mathsf{sk}_k) \leftarrow \mathsf{KeyShareGen}$
   b) Send $(\mathsf{pk}_k, \mathsf{sk}_k)$ to T_k and pk_k to Sim

Compute: On input $(\texttt{Compute}, \mathsf{pk}_k, \mathsf{sk}_k)$ from each $\mathsf{T}_k \in \mathsf{T}$ and input $(\texttt{Compute}, (c_i^{\mathsf{choice}} \in \mathfrak{C})_{i=1}^{n_{\mathsf{voters}}}$ from BB:

1. $\forall i \in \{1, \ldots, n_{\mathsf{voters}}\}$:
   a) $\forall k \in \{1, \ldots, n_{\mathsf{trustees}}\}\colon \mathsf{dec}_{\mathsf{sk}_k}(i) \leftarrow \mathsf{dec}_{\mathsf{sk}_k}(E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_i)$
   b) $c_i^{\mathsf{choice}} \leftarrow \mathsf{dec}(\mathsf{dec}_{\mathsf{sk}_1}(i), \ldots, \mathsf{dec}_{\mathsf{sk}_{n_{\mathsf{trustees}}}}(i))$
2. Return res $\leftarrow f^{\mathsf{MPC}}(f^{\mathsf{agg}}((c_i^{\mathsf{choice}})_{i=1}^{n_{\mathsf{voters}}}))$ to Sim.

Figure 3.8.: Ideal MPC protocol.

Now, the privacy theorem for Ordinos says that the level of privacy of Ordinos for this class of adversaries is the same as the one for the ideal protocol with $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ honest voters.

Theorem 3.4 (Privacy of Ordinos)**.** *Under the assumptions (P1) to (P4) and with the mapping* $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}})$ *as defined above,* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, \mathfrak{C}, f^{\mathrm{res}})$ *achieves a privacy level of* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ *w.r.t.* $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}})$.

Proof. The proof is provided in [Mül19], where we reduce the privacy game for Ordinos with $n_{\mathsf{voters}}^{\mathsf{honest}}$ honest voters, as specified in Definition 2.3, to the privacy game for the ideal voting protocol with $n_{\mathsf{voters}}^{\mathsf{honest}} - k$ voters, by a sequence of games. □

Remark 3.1. Recall that in Ordinos, the trustees evaluate the election result function f^{res} over the homomorphically aggregated votes, i.e., the vector that encrypts the total number of votes for each (choice) component. Conversely, the more general result function of the ideal voting protocol receives the voters' choices as input. Hence, this generalized result function must aggregate the votes to apply f^{res}.

Since the risk of being caught cheating increases exponentially with k, the number of changed votes k will be relatively small in practice. Nevertheless, the privacy theorem suggests that manipulating just a few votes of honest voters does not buy the adversary much to weaken privacy, as illustrated in Section 3.1.3. Conversely, the result function can very well affect the level of privacy of a tally-hiding system: whether we only announce the winner of an election or the complete result typically significantly affects privacy. As the following theorem states, Ordinos achieves full tally-hiding.

Theorem 3.5 (Full Tally-Hiding of Ordinos)**.** *Under the assumptions (P1) to (P4) and with the mapping* $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}})$ *as defined above,* $\mathsf{P}_{\mathsf{Ordinos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, p_{\mathsf{verify}}, p_{\mathsf{audit}}, \mathfrak{C}, f^{\mathrm{res}})$ *achieves full tally-hiding w.r.t.* (f^{res}) *and* (T, t)*.*

Proof. This theorem directly follows Theorem 3.4. □

3.2.3. Instantiations

This section presents our instantiations of the Ordinos framework. We must specify multiple components to instantiate the Ordinos framework. As cryptographic primitives, we need a homomorphic, IND-CPA-secure $(t, n_{\mathsf{trustees}})$*-threshold public-key encryption scheme* defined as $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, E_{\mathsf{pk}}, \mathsf{dec}^{\mathsf{share}}, \mathsf{dec})$, such as exponential ElGamal or Paillier. A *non-interactive zero-knowledge proof (NIZKP)* $\pi^{\mathfrak{C}}$ for proving knowledge and correctness of a plaintext vector given ciphertext vector, a public key, and a choice space $\mathfrak{C}$; a NIZKP $\pi^{\mathsf{KeyShareGen}}$ for proving knowledge and correctness of a private key share given a public key share (see Appendix A.4 for details). A *multi-party computation (MPC) protocol* $\mathsf{P}^{f^{\mathrm{res}}}$ that takes as input a ciphertext vector of encrypted integers (encrypted using $\mathcal{E}$ from above) and securely evaluates a given function f^{res} over the plain integers and outputs the result on a bulletin board. For example, we can use the function $f^{\mathrm{res}}_{\mathsf{Plurality}}$ that outputs the index(s) of the ciphertext(s) with the highest integer to determine and publish the winner of an election, see Section 2.7. We precisely define the exact security properties $\mathsf{P}^{f^{\mathrm{res}}}$ has to satisfy to achieve privacy and verifiability/accountability for Ordinos in Section 3.2.2. Our work's main challenge and core contribution is demonstrating that these components are suitable for constructing protocols for simple and complex voting methods and result functions. Finally, one needs an EUF-CMA-secure *signature scheme* $\mathcal{S}$.

We obtained all our benchmarks using an ESPRIMO Q957 (64 bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM). We do not use parallelism; we use just a single core to make the results independent of the specific core count, making it easier to compare with other benchmarks.

3.2.3.1. Existing Cryptographic Primitives

This section presents how we instantiate the cryptographic primitives needed for the Ordinos framework.

We use a threshold variant of the Paillier encryption scheme [DJN10] with a key size of $2,048$ bits to implement $\mathcal{E}$.

The works of Nishide and Sakurai [NS10] (refined and implemented by Veugen et al. [VAS19]) and Hazay et al. [HMR+19] present secure distributed key generation for the Paillier cryptosystem (and these works also define $\pi^{\mathsf{KeyShareGen}}$):

- The MPC protocol proposed by Nishide and Sakurai [NS10] employs verifiable secret sharing with Pedersen commitments. The parties of the MPC protocol prove that they behaved correctly at every protocol step using zero-knowledge proofs proofs for committed values.
- In the MPC protocol proposed by Hazay et al. [HMR+19], the parties use a shared key for exponential ElGamal for joint computations and engage in two-party protocols based on the Paillier cryptosystem for each pair of parties. The parties employ zero-knowledge proofs to ensure that encrypted values are chosen and used correctly in each step.

The main difference between both proposals is that Nishide and Sakurai's protocol is secure in the malicious setting with an honest majority, while Hazay et al. present a solution secure against a dishonest majority. Specifically, [NS10] is secure against the corruption of all but one of the parties. To securely create distributed key shares for a Paillier encryption scheme with a key size of $2,048$ bits, the protocol of [NS10, VAS19] requires about 15 minutes, while the protocol of [HMR+19] requires only about a single minute. We implemented the protocol of Nishide and Sakuarai [NS10, VAS19] for our Ordinos instantiations.

As for the signature scheme $\mathcal{S}$, we can use any EUF-CMA-secure instantiation. We make use of DSS [NIS23] for Ordinos.

3.2.3.2. Choice Space NIZKPs

We realize NIZKPs of choice spaces using existing NIZKPs from the literature. For example, the works of [CDS94, SV15] provide NIZKPs for $\mathfrak{C}_{\mathsf{Single}}$, which can also applied in the setting of $\mathfrak{C}_{\mathsf{DuelMatrix}}$, $\mathfrak{C}_{\mathsf{multi},=,n_{\mathsf{votes}},b}$, and $\mathfrak{C}_{\mathsf{RankingPermutation}}$. Furthermore, the work of [HPT19] presents a NIZKP for $\mathfrak{C}_{\mathsf{RankingMatrix}}$.

3.2.3.3. Existing Building Blocks of $\mathsf{P}^{f^{\mathrm{res}}}$

We have chosen the components presented in this section not only because they meet the necessary security requirements but also due to their efficiency, which facilitates constructing practical instantiations.

As stated above, we use a threshold variant of the additively homomorphic Paillier encryption scheme [DJN10] with a key size of $2,048$ bits to implement $\mathcal{E}$. Given a public key pk for this encryption scheme, we distribute the secret key among the n_{trustees} trustees, and at least t many trustees need to cooperate in order to decrypt a ciphertext. We can compute several functions over the plaintexts solely based on the ciphertexts due to the homomorphic property

of the Paillier encryption scheme. While we can compute basic operations locally, i.e., without interaction between the trustees, more complex operations need the trustees to work together. They do so by running publicly accountable MPC building blocks that require the participation of the owners of the corresponding secret key shares. In the following, we present these basic and more complex operations for $a, b, c \in \mathbb{Z}_n$ (all operations are mod n where n is determined by pk):

Basic Arithmetic Operations. Using the additive homomorphic property of the Paillier encryption scheme [Pai99], we can locally, without any interaction with other key shareholders, perform additions based on the ciphertexts: $E_{\mathsf{pk}}(c) = f_{\mathsf{add}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = a + b$; for brevity we denote this operation by $\oplus$. Additionally, the Paillier encryption scheme supports local multiplication with a publicly known constant: $E_{\mathsf{pk}}(c) = f_{\mathsf{mulpub}}(E_{\mathsf{pk}}(a), b)$ s.t. $c = a \cdot b$.

Shared Decryption. The most simple operation that requires interaction between the trustees that hold the secret key shares is the shared decryption: $c = f_{\mathsf{dec}}(E_{\mathsf{pk}}(c))$ s.t. $E_{\mathsf{pk}}(c)$ is an encryption of c. This operation is performed by letting each trustee T_i creating a decryption share $\mathsf{dec}^{\mathsf{share}}_{i,E_{\mathsf{pk}}(c)}$ by running $\mathsf{dec}^{\mathsf{share}}(\mathsf{sk}_i, c)$. Additionally, the trustee creates a NIZKP of correct decryption. We can reconstruct the plaintext c using at least t many such decryption shares.

Multiplication. The shared decryption is used as a sub-step to realize the following operations. In particular, using shared decryption, it is possible to multiply two plaintexts based on their ciphertexts, that is, $E_{\mathsf{pk}}(c) = f_{\mathsf{mul}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = a \cdot b$, for brevity we denote this operation by $\odot$. This protocol is described [DJN10].

Comparisons. More complex functions can be crafted using the operations presented above. For example, Lipmaa and Toft [LT13] realize comparisons. That is, a greater-than test $E_{\mathsf{pk}}(c) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = 1$ if (and only if) $a \geq b$ and 0 otherwise, and a test for equality $E_{\mathsf{pk}}(c) = f_{\mathsf{eq}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(b))$ s.t. $c = 1$ iff $a = b$ and 0 otherwise. These comparison protocols offer sublinear runtime as long as an upper bound $< n$ for both input values a and b is known; hence, performance drastically increases as long as a, b are known to remain small. With smaller values of n, these comparison protocols become faster because the length of encrypted integers to be compared by $f_{\mathsf{gt}}()$ determines the number of recursive calls of $f_{\mathsf{gt}}()$ from [53]. In essence, this protocol splits the inputs into an upper and lower half and recursively calls itself using one of those halves, depending on the output of the previous comparison. Hence, we use powers of 2 for the bit length of the integers. On a high level, this is also the reason for the logarithmic online complexity of $f_{\mathsf{gt}}()$. In contrast, $f_{\mathsf{eq}}()$ does not use recursion. We note that $f_{\mathsf{gt}}()$ and $f_{\mathsf{eq}}()$ can, in principle, also be used with exponential ElGamal; both functions use decryption for an (upper-bounded but still) relatively large plaintext space and, hence, would perform poorly with exponential ElGamal.

Figure 3.9 presents benchmarks of $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$, $f_{\mathsf{eq}}()$, $f_{\mathsf{gt}}()$ for values of different bit sizes, with five trustees communicating over a local network and a threshold of two. Because of the recursive protocols, bit sizes that are not powers of 2 have the same runtime as the next power of 2. The benchmarks demonstrate the independence of the runtime of $f_{\mathsf{dec}}()$ and $f_{\mathsf{mul}}()$ to the

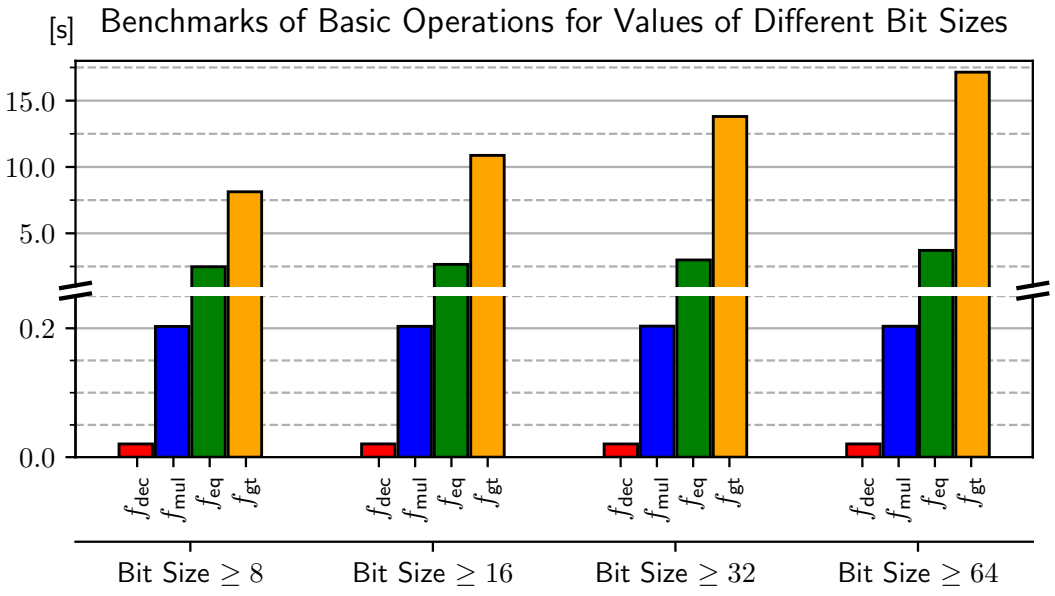


Figure 3.9.: Benchmarks of $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$, $f_{\mathsf{eq}}()$, $f_{\mathsf{gt}}()$.

bit size. The runtime of $f_{\mathsf{eq}}()$ barely increases with increasing bit size. Only the benchmarks of $f_{\mathsf{gt}}()$ are majorly affected by the bit size.

The bit size of the values determines how many voters the instantiation of Ordinos supports: The values of the aggregated votes per component must not exceed the maximum supported value. Thus, with values of 16 bits and $\mathfrak{C}_{\mathsf{Single}}$, the Ordinos instantiation supports up to 2^{16} voters. These numbers might differ for other choice spaces, as voters can give more than one vote to a component, for example, for $\mathfrak{C}_{\mathsf{Multi},\sigma_{\mathsf{comp}},n_{\mathsf{votes}},b}$ or $\mathfrak{C}_{\mathsf{BordaPointList}}$.

Figure 3.10 presents benchmarks of $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$, $f_{\mathsf{eq}}()$, $f_{\mathsf{gt}}()$ using different thresholds of the encryption scheme. We employed twelve trustees to communicate over a local network for these benchmarks. We used three systems described above, each running four trustees, one per core. The threshold barely impacts the runtimes of $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$: To conduct a decryption or a multiplication, a trustee requires input from $t-1$ many other trustees and must verify the received messages, which, incorporates verifying ZKPs. Consequently, raising the threshold increases the computation complexity of each trustee. The protocols $f_{\mathsf{eq}}()$ and $f_{\mathsf{gt}}()$ utilize $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$ multiple times as sub-protocols; hence the threshold also influences their runtime. Nevertheless, the increase in the runtime is diminutive and mainly noticeable for $f_{\mathsf{gt}}()$ and more significant thresholds.

Our electronic voting system requires not only MPC protocols with appropriate computational complexity but also optimized communication between the trustees. This is because the trustees communicate over a network during the MPC protocol, and we need to minimize communication between them to avoid slowing down the overall computation due to slow connections. The sublinear comparison protocols developed by Lipmaa and Toft [LT13] not only offer sublinear computation complexity but are also optimized in terms of communication complexity.

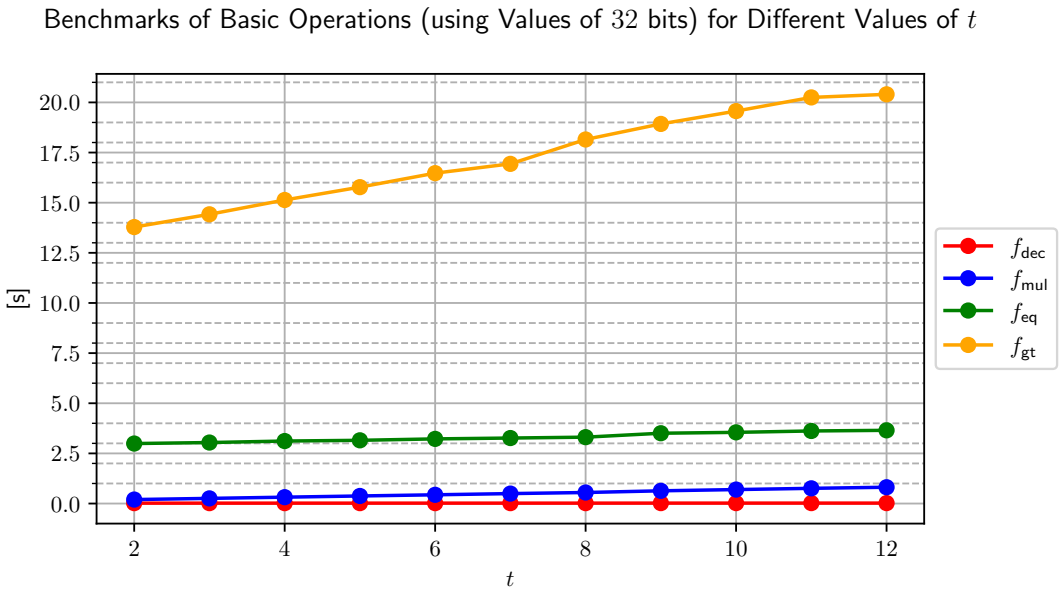


Figure 3.10.: Benchmarks of $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$, $f_{\mathsf{eq}}()$, $f_{\mathsf{gt}}()$ for various values of t.

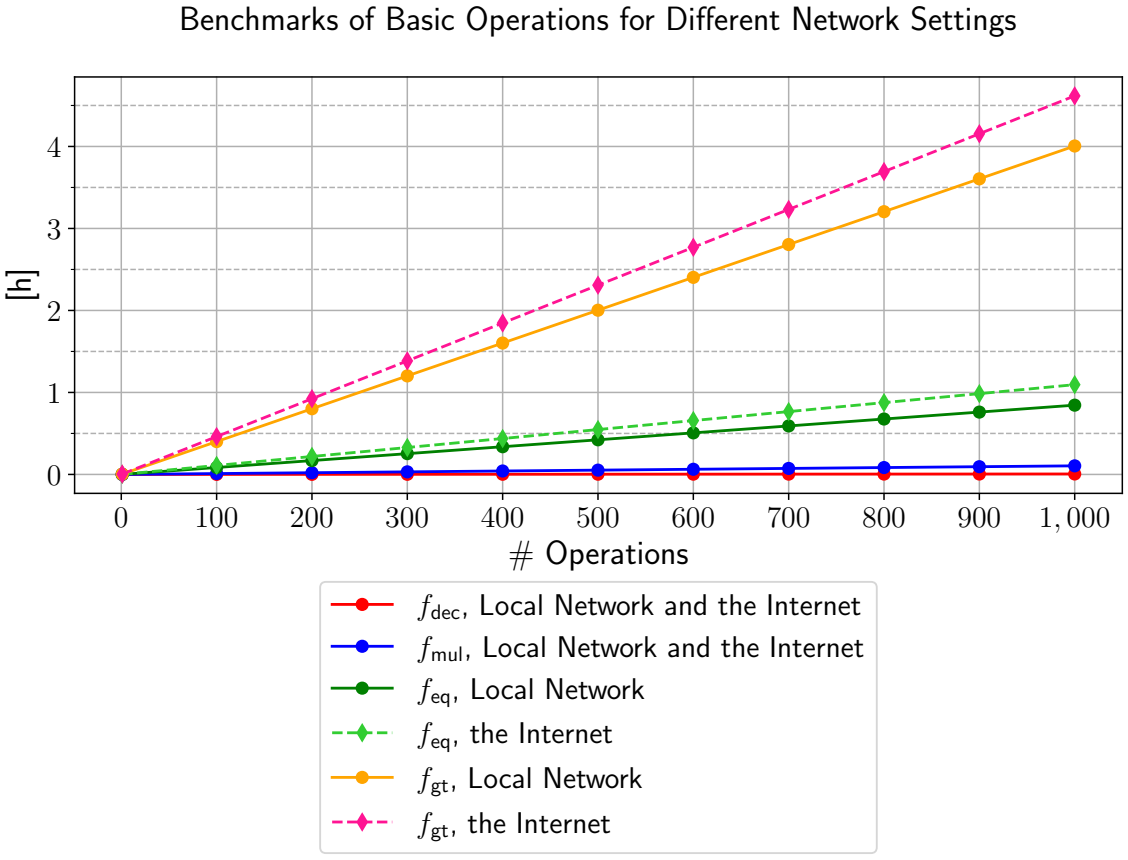


Figure 3.11.: Benchmarks of $f_{\mathsf{dec}}()$, $f_{\mathsf{mul}}()$, $f_{\mathsf{eq}}()$, $f_{\mathsf{gt}}()$ over a local network and the Internet, three trustees and $t = 3$.

We conducted runtime measurements of the MPC building blocks in Figure 3.11. For this, we set up two communication channels – one over a local network and another over the Internet. The differences in runtime are relatively minor: performing $1,000$ greater-than tests increases the overall runtime by only 12% when switching from a local network to the Internet. This difference decreases for equality tests, and there is almost no noticeable difference for multiplications and decryptions.

The MPC building blocks for computing the above operations have a valuable property. Namely, we can use encrypted outputs from one building block as inputs for another such that the resulting combined protocol is still a secure, publicly accountable MPC protocol [LT13]. In other words, they allow for building more complex protocols such as $\mathsf{P}^{f^{\text{res}}}$ for Ordinos that meet the requirements of Theorems 3.2, 3.4, and 3.5.

Secure Offline Phase. The building blocks presented above, the building blocks presented next, and our MPC protocols often require ciphertexts that contain specific values, e.g., a random value chosen uniformly and another ciphertext with the inverse of this random value as plaintext. Before the election, we compute such ciphertexts, with plaintexts independent of the inputs of the MPC protocol (in our case, the tally). We securely compute these ciphertexts for various plaintexts with specific properties using MPC protocols proposed in [CFL83a, CFL83b, BB89, DFK+06]. These works cover all of our use cases. The runtime of these MPC is not time-critical since we can execute them before the election.

3.2.3.4. Novel Building Blocks of $\mathsf{P}^{f^{\text{res}}}$

This section describes new publicly verifiable and accountable MPC building blocks needed to construct $\mathsf{P}^{f^{\text{res}}}$ for various election result functions f^{res}, based on the primitives and existing building blocks introduced above.

Before we present the multi-party computation protocols for specific election result functions, we present various protocols serving as building blocks.

Comparison Matrix. A valuable tool for many of the following MPC components when comparing n values $(a_i)_{i=1}^n$ is a comparison matrix $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)$ of size $n \times n$, which denotes in each entry $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{ij}$ whether value a_i is greater than (or equal to) value a_j, indicated by $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{ij} = E_{\mathsf{pk}}(1)$ if this is the case, and $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{ij} = E_{\mathsf{pk}}(0)$ otherwise. The MPC component $\mathsf{P}^{\text{ComparisonMatrix}}$ to compute $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)$ is presented in Algorithm 3.1. Since this algorithm compares each pairing of the candidates, it has a quadratic runtime in the number of candidates.

Comparison Vector. Based on the comparison matrix consisting of encryptions of zero and one, we can compute a ciphertext for each a_i that contains the number of comparisons that this entry has won, i.e., how many $j \neq i$ exist with $a_i \geq a_j$. Algorithm 3.2 presents $\mathsf{P}^{\text{ComparisonVector}}$ that computes the comparison vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(V)$ of size n such that $E_{\mathsf{pk}}^{\mathsf{Vector}}(V)_i$ denotes against how many other elements in $(a_i)_{i=1}^n$ the element a_i wins (or ties) a direct comparison. Since this algorithm calls $\mathsf{P}^{\text{ComparisonMatrix}}$ as a sub-routine, it also has a quadratic runtime.

Algorithm 3.1: $\mathsf{P}^{\text{ComparisonMatrix}}$

Input: $(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i)_{i=1}^n$

$E_{\mathsf{pk}}^{\mathsf{Matrix}}(C) = \text{CreateEncMatrix}(1, n, n)$
for $i \in \{1, \ldots, n\}$ **do**
  for $j \in \{i+1, \ldots, n\}$ **do**
    $E_{\mathsf{pk}}(g) = f_{\mathsf{gt}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i, E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_j)$
    $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{ij} = E_{\mathsf{pk}}(g)$ ▷ $C_{ij} = a_i \geq a_j$
    $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{ji} = E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g) \oplus f_{\mathsf{eq}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i, E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_j)$ ▷ $C_{ji} = a_j \geq a_i$
return $E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)$

Algorithm 3.2: $\mathsf{P}^{\text{ComparisonVector}}$

Input: $(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i)_{i=1}^n$

$E_{\mathsf{pk}}^{\mathsf{Matrix}}(C) = \mathsf{P}^{\text{ComparisonMatrix}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(a))$
$E_{\mathsf{pk}}^{\mathsf{Vector}}(V) = (E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{i1} \oplus \cdots \oplus E_{\mathsf{pk}}^{\mathsf{Matrix}}(C)_{in})_{i=1}^n$
return $E_{\mathsf{pk}}^{\mathsf{Vector}}(V)$

Minimum and Maximum Values. Repeatedly, we have a vector $E_{\mathsf{pk}}^{\mathsf{Vector}}(a)$ of n elements and want to find the k indices of largest (or smallest) values. That is, we want to compute n ciphertexts $E_{\mathsf{pk}}^{\mathsf{Vector}}(b)$ such that $b_i = 1$ if a_i is one the k largest (or smallest) values in $(a_i)_{i=1}^n$ and $b_i = 0$ otherwise. We can do so by making use of $\mathsf{P}^{\text{ComparisonVector}}$ presented above and then applying $f_{\mathsf{gt}}()$ to compare the ciphertexts of the comparison vector (containing the results for a_i) with a ciphertext on the number $n - k$ and obtain $E_{\mathsf{pk}}^{\mathsf{Vector}}(b)_i$. If there are multiple a_i with the same value, there might be more than k entries b_i that are 1. In cases where exactly k such values are required, one can use a tie-breaking mechanism such as the one described in [CGY22]. Analogously, we can proceed in order to find the smallest k values. Note that this algorithm can also be applied if k is not publicly known but only available as a ciphertext; in this situation, k is also not revealed by the algorithm. We use this property later in the context of the Hare-Niemeyer method. We denote these algorithms for computing the vectors $E_{\mathsf{pk}}^{\mathsf{Vector}}(b)_i$ by P^{Maxk}, (or P^{Mink}) presented in Algorithm 3.3, (or Algorithm 3.4). These algorithms have runtime $\mathcal{O}(n^2)$.

(Index of) Maximum. If we are just interested in obtaining a ciphertext $E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i$ of the maximum value a_i in the vector $(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i)_{i=1}^n$, we can do so more efficiently in linear runtime. That is, we start with the possible maximum $m = E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_0$ and iterate through all a_i's. For each a_i we test whether it is greater than the current maximum with $E_{\mathsf{pk}}(g) = f_{\mathsf{gt}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i, E_{\mathsf{pk}}(m))$ and adapt the maximum accordingly with $E_{\mathsf{pk}}(m) = g \odot E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g)) \odot E_{\mathsf{pk}}(m)$. This MPC component $\mathsf{P}^{\text{MaxVal}}$ iterates through all elements and stores the current maximum value. We can compute the minimum $\mathsf{P}^{\text{MinVal}}$ accordingly. Extending this algorithm allows finding an index of the largest (or smallest) value. The MPC component presented in Algorithm 3.6 uses the same approach as $\mathsf{P}^{\text{MaxVal}}$ but also stores the largest value's index. With that, if multiple

Algorithm 3.3: P^{Maxk}

Input: $l = (E^{\mathsf{Vector}}_{\mathsf{pk}}(a)_i)_{i=1}^n$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(V) = \mathsf{P}^{\text{ComparisonVector}}(l)$

$E_{\mathsf{pk}}(g') = E_{\mathsf{pk}}(n) \ominus E_{\mathsf{pk}}(k) \ominus E_{\mathsf{pk}}(1)$

return $(f_{\mathsf{gt}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(V)_i, E_{\mathsf{pk}}(g')))_{i=1}^n$

Algorithm 3.4: P^{Mink}

Input: $l = (E^{\mathsf{Vector}}_{\mathsf{pk}}(a)_i)_{i=1}^n, E_{\mathsf{pk}}(k)$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(V) = \mathsf{P}^{\text{ComparisonVector}}(l)$

return $(E_{\mathsf{pk}}(1) \ominus f_{\mathsf{gt}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(V)_i, E_{\mathsf{pk}}(k)))_{i=1}^n$

such values of maximum value exist, the last index is returned. These algorithms have runtime $\mathcal{O}(n)$.

Floor Division. Given a ciphertext $E_{\mathsf{pk}}(a)$ of some $a \in \mathbb{N}$ and a ciphertext $E_{\mathsf{pk}}(b)$ for some $b \in \mathbb{N}_{>1}$, this algorithm, described in Algorithm 3.7, is used to compute a ciphertext $E_{\mathsf{pk}}(i)$ with $i = \lfloor \frac{a}{b} \rfloor$. The algorithm also requires a value $u \in \mathbb{N}$, s.t. $u \cdot b$ does not exceed the plaintext space size and $i \in \{0, \ldots, u\}$. The algorithm performs a binary search to find the correct result. Suppose we know the plaintext value of the divisor b. In that case, we can optimize the algorithm by replacing the multiplication $E_{\mathsf{pk}}(j) \odot E_{\mathsf{pk}}(b)$ with $f_{\mathsf{mulpub}}(E_{\mathsf{pk}}(j), b)$, allowing us to save an encrypted multiplication per bit of the input values.

Fractions. Election methods for parliamentary elections often employ divisions that produce fractions, an issue for encryption schemes, and MPC protocols that operate on natural numbers, such as those presented and employed in this thesis. One common approach [HHK+21a, CGY22] to deal with divisions is to multiply all values by the least common multiple of all divisors used in a computation such that divisions are always guaranteed to produce natural numbers. Scaling can drastically increase the size of numbers, severely reducing the efficiency gain of the sublinear comparisons protocols $f_{\mathsf{gt}}()$ and $f_{\mathsf{eq}}()$. Therefore, we propose an alternative approach to deal with fractions by representing values, where needed, as rational numbers. Let a be a rational number consisting of a numerator n and denominator d. We denote an encrypted rational number as $E^{\mathsf{Frac}}_{\mathsf{pk}}(a) := (E_{\mathsf{pk}}(n), E_{\mathsf{pk}}(d))$, and with $E^{\mathsf{Frac}}_{\mathsf{pk}}(a).\mathsf{num}$ we denote the encrypted numerator and with $E^{\mathsf{Frac}}_{\mathsf{pk}}(a).\mathsf{den}$ the encrypted denominator of a. We denote by CreateEncFraction(n, d) the operation that creates an encrypted rational number with numerator n and denominator d (if the inputs n or d are not already encrypted, then they are first encrypted with public constant randomness). Based on this representation, we design and implement MPC components for basic arithmetic computations on encrypted rational numbers, including addition, multiplication, and comparisons.

Based on $\mathsf{P}^{\text{MaxLastIdx}}$ (see Algorithm 3.6), we propose the protocol $\mathsf{P}^{\text{MaxFractionLastIdx}}$ that takes a list of n encrypted fractions and returns another list of the same length with $E_{\mathsf{pk}}(1)$

Algorithm 3.5: $\mathsf{P}^{\mathrm{MaxVal}}$

Input: $(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i)_{i=1}^n$
Result: $E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i$ such that $a_i = \max((a_i)_{i=1}^n)$

$E_{\mathsf{pk}}(m) = E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_1$ ▷ m contains the (current) maximum value
for $j \in \{2, \ldots, n\}$ **do**
  $E_{\mathsf{pk}}(g) = f_{\mathsf{gt}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_j, E_{\mathsf{pk}}(m))$
  $E_{\mathsf{pk}}(m) = E_{\mathsf{pk}}(g) \odot E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_j \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g)) \odot E_{\mathsf{pk}}(m)$ ▷ Update m
return $E_{\mathsf{pk}}(m)$

Algorithm 3.6: $\mathsf{P}^{\mathrm{MaxLastIdx}}$

Input: $(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_i)_{i=1}^n$
Result: Index of maximum value

$E_{\mathsf{pk}}(m) = E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_0$ ▷ m contains the (current) maximum value
$E_{\mathsf{pk}}(i) = E_{\mathsf{pk}}(0)$ ▷ i contains the index of the maximum value
for $j \in \{2, \ldots, n\}$ **do**
  $E_{\mathsf{pk}}(g) = f_{\mathsf{gt}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_j, E_{\mathsf{pk}}(m))$
  $E_{\mathsf{pk}}(m) = E_{\mathsf{pk}}(g) \odot E_{\mathsf{pk}}^{\mathsf{Vector}}(a)_j \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g)) \odot E_{\mathsf{pk}}(m)$ ▷ Update m
  $E_{\mathsf{pk}}(i) = E_{\mathsf{pk}}(g) \odot E_{\mathsf{pk}}(j) \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g)) \odot E_{\mathsf{pk}}(i)$ ▷ Update i
return $E_{\mathsf{pk}}(i)$

at the index of the maximal fraction and $E_{\mathsf{pk}}(0)$ everywhere else, where if there are multiple maxima, only the last one in the list is marked $E_{\mathsf{pk}}(1)$.

Election methods often need to deal with breaking ties. For this purpose, Cortier et al. [CGY22] proposed an algorithm that finds the maximum in a list and takes care of tie-breaking by scaling values and adding small tie-breaking values. While this scaling idea is conceptually simple, it requires care to obtain a correct implementation. For example, in the method proposed by Cortier et al., we compute a score for each c_j^{party} as $m_j = 2^{\lceil \log n_{\mathsf{parties}} + 1 \rceil} \cdot q_j + r_j$. We then compare the values q_j. Here, $r_j \in \{1, \ldots, n_{\mathsf{parties}}\}$ denotes a tie-breaking value (or *rank*) uniquely assigned to c_j^{party}. The values q_j are the values that one would compare without tie-breaking (if $q_i > q_j$ then party i should always win this comparison). However, suppose the values q_j are fractions of very close values. In that case, one can construct examples where this tie-breaking approach fails, i.e., $m_i > m_j$ but $q_i < q_j$. In this situation, we consider the ranks, although the values q_i and q_j are different; hence, we require no tie-breaking. These cases can occur in applying the Sainte-Laguë method. For this reason, we present a new protocol $\mathsf{P}^{\mathrm{MaxFractionByRank}}$ for breaking ties that does not run into this issue. This algorithm additionally takes encrypted ranks $E_{\mathsf{pk}}^{\mathsf{Vector}}(r) = (E_{\mathsf{pk}}(r_i))_{i=1}^n$ as input, where the $(r_1, \ldots, r_n)$ form a permutation of $0, \ldots, n-1$, and first scales all ciphertexts $E_{\mathsf{pk}}^{\mathsf{Vector}}(q)_i$ by a certain value, adds the encrypted ranking $E_{\mathsf{pk}}^{\mathsf{Vector}}(r)_i$ to the scaled $E_{\mathsf{pk}}^{\mathsf{Vector}}(q)_i$, and then continues just as $\mathsf{P}^{\mathrm{MaxFractionLastIdx}}$. By scaling, adding r_i does not change the output if the q_j are not tied. However, if any of the inputs q_j are equal, the

Algorithm 3.7: $\mathsf{P}^{\text{FloorDiv}}$

Input: $E_{\mathsf{pk}}(a)$, $E_{\mathsf{pk}}(b)$, u

$length = \text{BITLENGTH}(u)$
$E_{\mathsf{pk}}(lower) = E_{\mathsf{pk}}(0)$
for $i \in \{1, \ldots, length\}$ **do**
  $E_{\mathsf{pk}}(j) = E_{\mathsf{pk}}(lower) \oplus E_{\mathsf{pk}}(2^{length-i})$
  $E_{\mathsf{pk}}(gt) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(a), E_{\mathsf{pk}}(j) \odot E_{\mathsf{pk}}(b))$ ▷ Check if $a \geq j \cdot b$
  $E_{\mathsf{pk}}(lower) = E_{\mathsf{pk}}(lower) \oplus (2^{length-i} \odot E_{\mathsf{pk}}(gt))$ ▷ Update *lower* accordingly
return $E_{\mathsf{pk}}(lower)$

Algorithm 3.8: $\mathsf{P}^{\text{MaxFractionByRank}}$

Input: $(E_{\mathsf{pk}}^{\mathsf{FracVector}}(v)_i)_{i=1}^n$, $(E_{\mathsf{pk}}^{\mathsf{Vector}}(r)_i)_{i=1}^n$

$E_{\mathsf{pk}}^{\mathsf{Frac}}(v_{\max}) = E_{\mathsf{pk}}^{\mathsf{FracVector}}(v)_1$
$E_{\mathsf{pk}}(i_{\text{idx}}) = E_{\mathsf{pk}}(1)$
$E_{\mathsf{pk}}(r_{\max}) = E_{\mathsf{pk}}^{\mathsf{Vector}}(r)_1$
for $i \in \{2, \ldots, n\}$ **do**
  $E_{\mathsf{pk}}(m_i) = E_{\mathsf{pk}}^{\mathsf{FracVector}}(v)_i \odot E_{\mathsf{pk}}^{\mathsf{FracVector}}(v)_i.\mathsf{den} \odot E_{\mathsf{pk}}^{\mathsf{Frac}}(v_{\max}).\mathsf{den} \odot n \oplus E_{\mathsf{pk}}^{\mathsf{Vector}}(r)_i$
  $E_{\mathsf{pk}}(m_{\max}) = E_{\mathsf{pk}}^{\mathsf{Frac}}(v_{\max}) \odot E_{\mathsf{pk}}^{\mathsf{Frac}}(v_{\max}).\mathsf{den} \odot E_{\mathsf{pk}}^{\mathsf{FracVector}}(v)_i.\mathsf{den} \odot n \oplus E_{\mathsf{pk}}(r_{\max})$
  $E_{\mathsf{pk}}(w) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(m_i), E_{\mathsf{pk}}(m_{\max}))$
  $E_{\mathsf{pk}}^{\mathsf{Frac}}(v_{\max}) = E_{\mathsf{pk}}(w) \odot E_{\mathsf{pk}}^{\mathsf{FracVector}}(v)_i \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(w)) \odot E_{\mathsf{pk}}^{\mathsf{Frac}}(v_{\max})$
  $E_{\mathsf{pk}}(i_{\text{idx}}) = E_{\mathsf{pk}}(w) \odot E_{\mathsf{pk}}(i) \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(w)) \odot E_{\mathsf{pk}}(i_{\text{idx}})$
  $E_{\mathsf{pk}}(r_{\max}) = E_{\mathsf{pk}}(w) \odot E_{\mathsf{pk}}^{\mathsf{Vector}}(r)_i \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(w)) \odot E_{\mathsf{pk}}(r_{\max})$
return $(f_{\mathsf{eq}}(E_{\mathsf{pk}}(i), E_{\mathsf{pk}}(i_{idx})))_{i=1}^n$

party with the highest rank r_i will have the greater (encrypted) value after the addition. We present the algorithm in Algorithm 3.8.

3.2.3.5. MPC Components for Election Result Functions

We instantiate the Ordinos framework with a deterministic polynomial election result function $f^{\text{res}} : \{0,1\} \to \{0,1\}$ that computes the final election result based on the (aggregated) tally (see Section 3.2.1).

Depending on the election result function f^{res}, the Ordinos system uses a publicly accountable MPC protocol $\mathsf{P}^{f^{\text{res}}}$ that computes f^{res} in a secure and tally-hiding way. That is, $\mathsf{P}^{f^{\text{res}}}$ takes as input the encrypted aggregated tally $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ and then computes the election result such that

$$\mathsf{P}^{f^{\text{res}}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})) = f^{\text{res}}(c_{\mathsf{agg}}^{\mathsf{choice}}).$$

This section presents various secure MPC instantiations $\mathsf{P}^{f^{\text{res}}}$ that evaluate f^{res}. Section 2.7 presents various election result functions. We note that the Ordinos framework currently supports

Algorithm 3.9: $\mathsf{P}^{f^{\text{res}}_{\text{Plurality}}}$

Input: $E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})$

1 $E_{\text{pk}}(v_{\max}) = \mathsf{P}^{\text{MaxVal}}(E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}}))$

2 **return** $(f_{\text{dec}}(f_{\text{eq}}(E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})_i, E_{\text{pk}}(v_{max}))))_{i=1}^{n_{\text{components}}}$

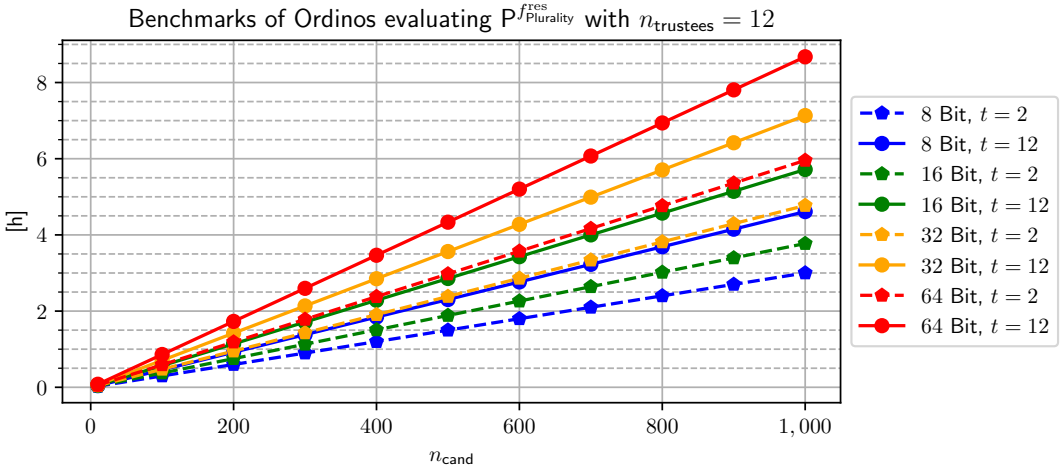


Figure 3.12.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{Plurality}}}$ with $n_{\text{trustees}} = 12$.

the result functions presented in this section, including several variants. At the end of this section, we prove security for all of these instantiations of Ordinos.

3.2.3.6. Plurality Voting

The $f^{\text{res}}_{\text{Plurality}}$ function outputs (choice) component with the most votes. The $\mathsf{P}^{f^{\text{res}}_{\text{Plurality}}}$ MPC protocol, defined in Algorithm 3.9, finds the maximum value of votes per (choice) component and checks for each (choice) component whether it received the most votes. Using $\mathsf{P}^{\text{MaxVal}}$ to find the maximum value leads to a linear runtime in n_{cand}.

Figure 3.12 presents the benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{Plurality}}}$ for various bit sizes and thresholds. We used 12 trustees for all benchmarks. The benchmarks validate that the runtime is linear in the number of candidates. Furthermore, they reveal that the bit size of the values (which determine the number of recursive calls) is the main factor contributing to the runtime. Regarding the threshold t, although the differences in runtime for basic operations are barely noticeable (see Figure 3.10), protocols for election result functions require multiple calls of these basic operations, resulting in evident differences in runtime.

Using Ordinos, we can evaluate $f^{\text{res}}_{\text{Plurality}}$ with 100 candidates in about 51 minutes, using values of 64 bits, 12 trustees, and a threshold of 12. Most elections do not require values of 64 bits, as they are larger than the number of voters in all cases. However, we could need values of larger bit sizes for elections that allow voters to assign points to candidates. Using 16 bits and

Algorithm 3.10: $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$

Input: $E^{\mathsf{Vector}}_{\mathsf{pk}}(c^{\mathsf{choice}}_{\mathsf{agg}})$

1 **return** $(f_{\mathsf{dec}}(f_{\mathsf{gt}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(c^{\mathsf{choice}}_{\mathsf{agg}})_i, t)))_{i=1}^{n_{\mathsf{cand}}}$

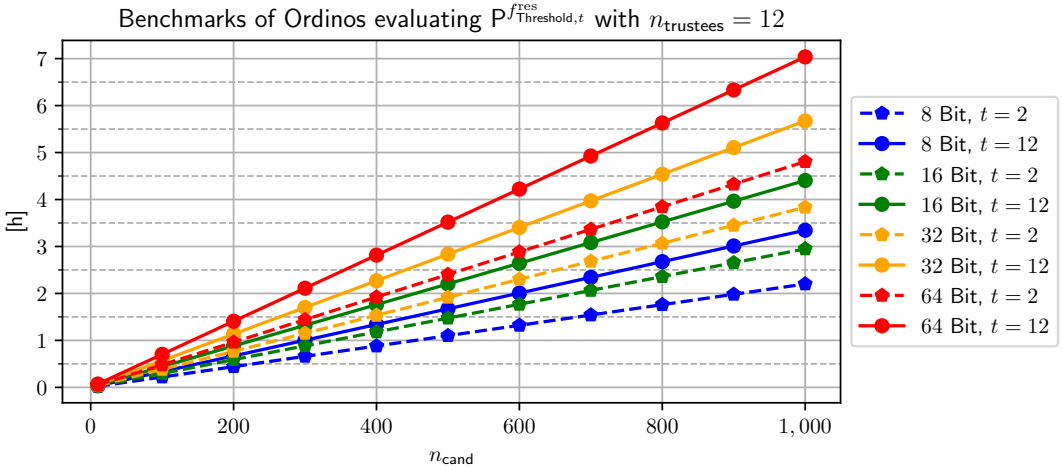


Figure 3.13.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$ with $n_{\mathsf{trustees}} = 12$.

maintaining the same threshold and number of trustees, we can evaluate a plurality vote with 100 candidates in approximately 34 minutes.

3.2.3.7. Threshold

The election result function $f^{\mathsf{res}}_{\mathsf{Threshold},t}$ outputs all candidates that received at least t many votes. We present the secure tally-hiding MPC realization $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$ in Algorithm 3.10. It computes $(f_{\mathsf{dec}}(f_{\mathsf{gt}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(c^{\mathsf{choice}}_{\mathsf{agg}})_i, E_{\mathsf{pk}}(t))))_{i=1}^{n_{\mathsf{cand}}}$ and outputs this list. Therefore, the complexity of this protocol is linear in the number of candidates and dominated by running $f_{\mathsf{gt}}()$ per candidate.

Figure 3.13 presents the benchmarks of Ordinos evaluating $f^{\mathsf{res}}_{\mathsf{Threshold},t}$ for various bit sizes and thresholds. As usual, we used 12 trustees for all benchmarks.

Using Ordinos, we can evaluate $f^{\mathsf{res}}_{\mathsf{Threshold},t}$ with 100 candidates in about 34 minutes, using values of 32 bits, 12 trustees, and a threshold of 12.

3.2.3.8. Ranking, Parital Ranking, Best, and Ranking with Aggregated Votes of Best

Our Ordinos instantiations for the election result functions $f^{\mathsf{res}}_{\mathsf{Ranking}}$, $f^{\mathsf{res}}_{\mathsf{PartialRanking},n}$, $f^{\mathsf{res}}_{\mathsf{Best},n}$, and $f^{\mathsf{res}}_{\mathsf{RankingVotesBest},n}$ are quite similar and only differ in minor aspects. Therefore, we will present them together in this section.

The core of the multi-party protocols for these instantiations is $\mathsf{P}^{\mathrm{ComparisonVector}}$. This building block essentially computes the ranking of the candidates, and our instantiation $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{Ranking}}}$ calls $\mathsf{P}^{\mathrm{ComparisonVector}}$ and decrypts the resulting vector $E^{\mathsf{Vector}}_{\mathsf{pk}}(V)$. We realize $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{PartialRanking},n}}$ by modifying $E^{\mathsf{Vector}}_{\mathsf{pk}}(V)$ in the following way before decryption. We check for each entry via a

Algorithm 3.11: $\mathsf{P}^{f^{\text{res}}_{\text{Best},n}}$

Input: $E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})$

1 $E^{\text{Vector}}_{\text{pk}}(l) = \mathsf{P}^{\text{Maxk}}((E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})), n)$

2 **return** $(f_{\text{dec}}(E^{\text{Vector}}_{\text{pk}}(l)_i))_{i=1}^{n_{\text{cand}}}$

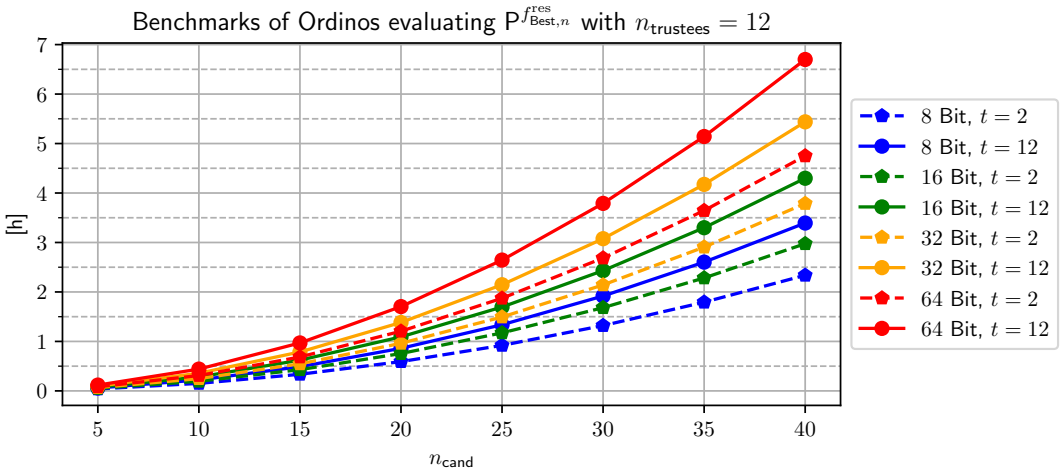


Figure 3.14.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{Best},n}}$ with $n_{\text{trustees}} = 12$. The benchmarks of $f^{\text{res}}_{\text{Ranking}}$, $f^{\text{res}}_{\text{PartialRanking},n}$, and $f^{\text{res}}_{\text{RankingVotesBest},n}$ only differ for up to a minute at maximum.

greater-than test $f_{\text{gt}}()$ whether this entry is at least n and set every entry that does not pass this check to zero using a multiplication $f_{\text{mul}}()$ for the respective tally-hiding assignment. The secure tally-hiding MPC realization $\mathsf{P}^{f^{\text{res}}_{\text{Best},n}}$ operates similarly but sets all values that pass the check to one and all others to zero before decryption.

The protocol $\mathsf{P}^{f^{\text{res}}_{\text{RankingVotesBest},n}}$ uses the same idea but does not make changes to the comparison vector $E^{\text{Vector}}_{\text{pk}}(V)$ but instead adapts the vote counts of each candidate. Using the above check, if the candidate is among the n candidates, her vote count is not updated; otherwise, we set it to zero. In addition to the ranking, the trustees decrypt the updated vote count.

We present $\mathsf{P}^{f^{\text{res}}_{\text{Best},n}}$ in Algorithm 3.11 and omit the formal description of the other protocols, due to their similarity to this protocol.

The protocols have similar runtimes that only differ up to a minute. We present the benchmarks of $\mathsf{P}^{f^{\text{res}}_{\text{Best},n}}$ in Figure 3.14 for various bit sizes and thresholds. As usual, we used 12 trustees for all benchmarks. The benchmarks show the quadratic scaling of the runtime in the number of candidates due to the underlying protocol $\mathsf{P}^{\text{ComparisonVector}}$.

Using Ordinos, we can evaluate the ranking-based election result function with 15 candidates in about 47 minutes, using values of 32 bits, 12 trustees, and a threshold of 12.

3.2.3.9. Condorcet

Condorcet methods are more complex than the election result function from the previous sections. The candidates do not receive votes that we aggregate; instead, the voters rank them with direct comparisons. We obtain an aggregated comparison matrix as input for the Condorcet election result functions, opening different and sophisticated mechanisms to select the election's winner(s).

Condorcet methods operate with computing scores for each candidate depending on her direct comparisons with the other candidates. Then, algorithms that select the election's winners based on these scores can be applied. For example, the Condorcet Schulze method creates a weighted graph between the candidates and searches for paths with optimal values.

These complex algorithms are demanding to compute for traditional elections, particularly for tally-hiding systems relying on heavy-weight cryptography. Therefore, a significant challenge for us is constructing full tally-hiding MPC protocols leading to efficient Ordinos instantiations.

3.2.3.10. Condorcet Helper

As input for the following Condorcet methods, we use the aggregated ranking matrix Ψ (see Section 2.6). To compute the comparison matrix and comparison vector, we need to adapt $\mathsf{P}^{\mathrm{ComparisonMatrix}}$ and $\mathsf{P}^{\mathrm{ComparisonVector}}$ accordingly to support the modified input. Therefore, we construct a helper method for the Condorcet methods, denoted as $\mathsf{P}^{\mathsf{CondorcetHelper}}$, that computes the *strict comparison matrix* D, which indicates in each entry D_{ij} whether candidate i has strictly won more comparisons with candidate j ($D_{ij} = 1$) or not ($D_{ij} = 0$). Furthermore, for each c_i^{cand}, the variable $\mathsf{d}_i^{\mathsf{gteq}}$ denotes the number of comparisons she has won or tied, while $\mathsf{d}_i^{\mathsf{gt}}$ only counts the winning comparisons. We present the MPC protocol in Algorithm 3.12.

3.2.3.11. Plain Condorcet

The vanilla/plain Condorcet method, $f^{\mathrm{res}}_{\mathsf{CondorcetPlain}}$, yields a unique winner who wins all pairwise comparisons. We present the tally-hiding realization $\mathsf{P}^{f^{\mathrm{res}}_{\mathsf{CondorcetPlain}}}$ in Algorithm 3.13. The MPC protocol searches the candidate that won $n_{\mathsf{cand}} - 1$ duels, i.e., won against all other candidates. For this, it employs $\mathsf{P}^{\mathsf{CondorcetHelper}}$ as a sub-protocol and then checks whether $\mathsf{d}_i^{\mathsf{gt}}$ equals $n_{\mathsf{cand}} - 1$.

Figure 3.15 presents the benchmarks of $\mathsf{P}^{f^{\mathrm{res}}_{\mathsf{CondorcetPlain}}}$. The benchmarks show the quadratic scaling of $\mathsf{P}^{\mathsf{CondorcetHelper}}$ in n_{cand}.

3.2.3.12. Weak Condorcet

In this method $f^{\mathrm{res}}_{\mathsf{CondorcetWeak}}$, all candidates that do not lose duels are output. Executing the algorithm for (plain) Condorcet allows a straightforward computation of the weak Condorcet winner. That is, for each candidate c_i^{cand} we test whether $f_{\mathsf{dec}}(f_{\mathsf{gt}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}})_i, E_{\mathsf{pk}}(n_{\mathsf{cand}} - 1))) = 1$. Algorithm 3.14 presents the tally-hiding realization.

Algorithm 3.12: $\mathsf{P}^{\mathsf{CondorcetHelper}}$

Input: $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})$

$E_{\mathsf{pk}}^{\mathsf{Matrix}}(D) = \mathrm{CreateEncMatrix}(0, n_{\mathsf{cand}}, n_{\mathsf{cand}})$ ▷ Strict comparison matrix

$E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}}) = \mathrm{CreateEncVector}(0, n_{\mathsf{cand}})$ ▷ Number of comparisons won or tied

$E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}}) = \mathrm{CreateEncVector}(0, n_{\mathsf{cand}})$ ▷ Number of comparisons won

for $i \in \{1, \ldots, n_{\mathsf{cand}}\}$ **do**

  for $j \in \{i+1, \ldots, n_{\mathsf{cand}}\}$ **do**

    $E_{\mathsf{pk}}(g) = f_{\mathsf{gt}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ij}, E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ji})$

    $E_{\mathsf{pk}}(e) = f_{\mathsf{eq}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ij}, E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ji})$

    $E_{\mathsf{pk}}(g') = E_{\mathsf{pk}}(g) \ominus E_{\mathsf{pk}}(e)$

    $E_{\mathsf{pk}}^{\mathsf{Matrix}}(D)_{ij} = E_{\mathsf{pk}}(g')$

    $E_{\mathsf{pk}}^{\mathsf{Matrix}}(D)_{ji} = E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g)$

    $E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}})_i = E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}})_i \oplus E_{\mathsf{pk}}(g)$

    $E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}})_j = E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}})_j \oplus E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g')$

    $E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}})_i = E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}})_i \oplus E_{\mathsf{pk}}(g')$

    $E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}})_j = E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}})_j \oplus E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g)$

return $E_{\mathsf{pk}}^{\mathsf{Matrix}}(D), E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}}), E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}})$

Algorithm 3.13: $\mathsf{P}^{f_{\mathsf{CondorcetPlain}}^{\mathrm{res}}}$

Input: $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})$

$E_{\mathsf{pk}}^{\mathsf{Matrix}}(D), E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gteq}}), E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}}) = \mathsf{P}^{\mathsf{CondorcetHelper}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}}))$

return $(f_{\mathsf{dec}}(f_{\mathsf{eq}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(\mathsf{d}^{\mathsf{gt}})_i, E_{\mathsf{pk}}(n_{\mathsf{components}} - 1))))_{i=1}^{n_{\mathsf{cand}}}$

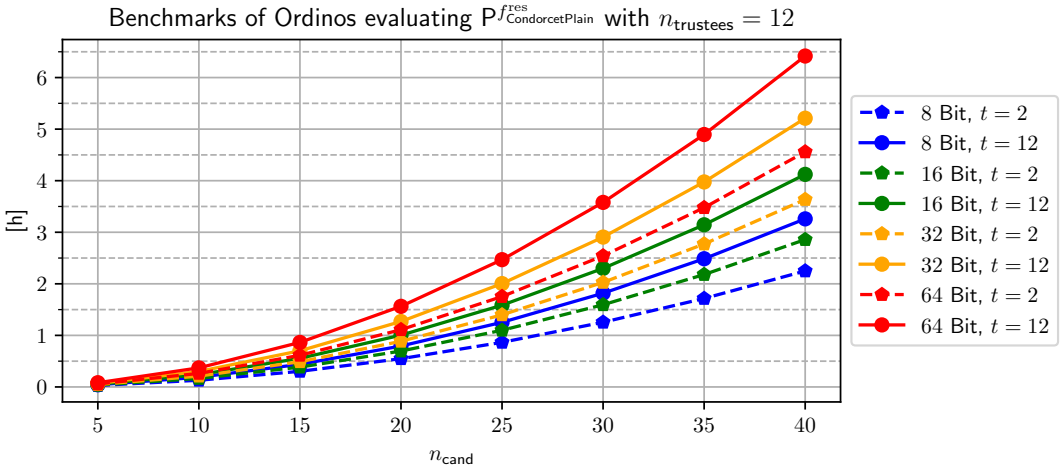


Figure 3.15.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f_{\mathsf{CondorcetPlain}}^{\mathrm{res}}}$ with $n_{\mathsf{trustees}} = 12$.

We present the benchmarks of $\mathsf{P}^{f_{\mathsf{CondorcetWeak}}^{\mathrm{res}}}$ in Figure 3.16, which are similar to the benchmarks of $\mathsf{P}^{f_{\mathsf{CondorcetPlain}}^{\mathrm{res}}}$.

Algorithm 3.14: $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetWeak}}}$

Input: $E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})$

1 $E^{\mathsf{Matrix}}_{\mathsf{pk}}(D), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}}), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gt}}) = \mathsf{P}^{\mathsf{CondorcetHelper}}(E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}}))$

2 **return** $(f_{\mathsf{dec}}(f_{\mathsf{eq}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}})_i, E_{\mathsf{pk}}(n_{\mathsf{cand}}-1))))^{n_{\mathsf{cand}}}_{i=1}$

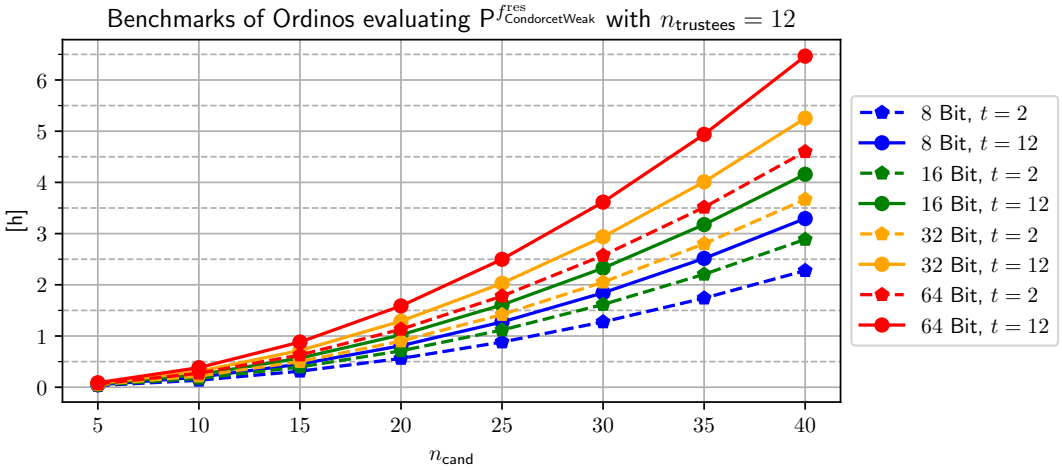


Figure 3.16.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetWeak}}}$ with $n_{\mathsf{trustees}} = 12$.

3.2.3.13. Copeland

This method $f^{\mathsf{res}}_{\mathsf{CondorcetCopeland}}$ considers the wins and losses of each candidate in their duels and outputs all candidates with the highest difference between wins and losses. For this evaluation method, we first compute the Copeland points of each candidate, which are defined as follows.

Definition 3.2 (Copeland Points)**.** *Let $C = \{c^{\mathsf{cand}}_1, \ldots, c^{\mathsf{cand}}_{n_{\mathsf{cand}}}\}$ be a set of candidates and let $n^1_{\mathsf{votes}}, \ldots, n^{n_{\mathsf{cand}}}_{\mathsf{votes}}$ be the number of votes that these candidates received. The Copeland points of c^{cand}_i, denoted as p^{Copeland}_i, are defined as follows. The candidate receives two points for every other candidate she wins the duel and one for each tied candidate. We formally define the Copeland points as*

$$p^{\mathsf{Copeland}}_i := \sum_{c^{\mathsf{cand}}_j \in C, c^{\mathsf{cand}}_j \neq c^{\mathsf{cand}}_i, n^i_{\mathsf{votes}} > n^j_{\mathsf{votes}}} 2 + \sum_{c^{\mathsf{cand}}_j \in C, c^{\mathsf{cand}}_j \neq c^{\mathsf{cand}}_i, n^i_{\mathsf{votes}} = n^j_{\mathsf{votes}}} 1.$$

Furthermore, let $S \subseteq C$. With p^{Copeland}_S we denote the sum of the Copeland points of the candidates in S, that is $p^{\mathsf{Copeland}}_S := \sum_{c^{\mathsf{cand}}_i \in S} p^{\mathsf{Copeland}}_i$.

The maximum Copeland points possible are $2 \cdot (n_{\mathsf{cand}} - 1)$ when the candidate wins the duel against every other candidate.

Using the results from the (plain) Condorcet evaluation, we compute the Copeland points of c^{cand}_i via $p^{\mathsf{Copeland}}_i = \mathsf{d}^{\mathsf{gt}}_i + \mathsf{d}^{\mathsf{gteq}}_i$. Since these comparisons do not include the comparison with the

Algorithm 3.15: $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetCopeland}}}$

Input: $E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})$
Result: Test
1 $E^{\mathsf{Matrix}}_{\mathsf{pk}}(D), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}}), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gt}}) = \mathsf{P}^{\mathsf{CondorcetHelper}}(E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}}))$
2 $E^{\mathsf{Vector}}_{\mathsf{pk}}(l) = (E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gt}})_i + E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}})_i)_{i=1}^{n_{\mathsf{cand}}}$ ▷ Compute Copeland points
3 $E_{\mathsf{pk}}(v_{\max}) = \mathsf{P}^{\mathrm{MaxVal}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(l))$
4 **return** $(f_{\mathsf{dec}}(f_{\mathsf{eq}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(l)_i, E_{\mathsf{pk}}(v_{max}))))_{i=1}^{n_{\mathsf{cand}}}$

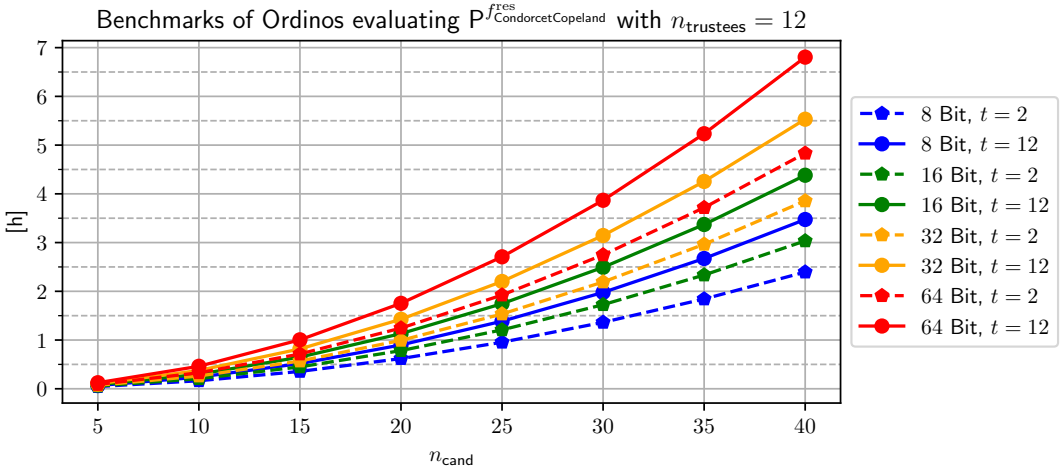


Figure 3.17.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetCopeland}}}$ with $n_{\mathsf{trustees}} = 12$.

candidate itself, the maximum Copeland points that a candidate can receive is $2 \cdot (n_{\mathsf{cand}} - 1)$, which occurs if the candidate strictly wins all of its $n_{\mathsf{cand}} - 1$ duels. The tally-hiding realization $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetCopeland}}}$ then computes the candidate with the most Copeland points with the help of the algorithm for the maximum. We present the algorithm in Algorithm 3.15.

Figure 3.17 presents the benchmarks of $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetCopeland}}}$, which are slightly slower as the runtime of $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{CondorcetPlain}}}$.

3.2.3.14. Smith Set

In this method $f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}$, a set of winners is output, the so-called *Smith set*. It is defined as the smallest set of candidates such that each candidate from the Smith set wins the duels against every candidate outside the Smith set. Otherwise, it will necessarily contain multiple candidates. In order to compute the Smith set in a tally-hiding way, we use the Copeland points (see Definition 3.2) to find the smallest dominating set, as defined below.

Definition 3.3 (Dominating Set)**.** *Let* $C = \{c^{\mathsf{cand}}_1, \ldots, c^{\mathsf{cand}}_{n_{\mathsf{cand}}}\}$ *be a set of candidates and let* $n^1_{\mathsf{votes}}, \ldots, n^{n_{\mathsf{cand}}}_{\mathsf{votes}}$ *be the number of votes that these candidates received. A subset* $\Phi \subseteq C$ *is called* dominating set *(of* C*) if (and only if) such that each candidate in* Φ *wins the direct duel against every candidate not in* Φ*, i.e., it holds that*

$$\forall c_i^{\mathsf{cand}} \in \Phi \, \forall c_j^{\mathsf{cand}} \in C \setminus \Phi : n_{\mathsf{votes}}^i > n_{\mathsf{votes}}^j.$$

Lemma 3.1 (Copeland Points of a Dominating Set)**.** *If $\Phi \subseteq C$ is a dominating set of C, then it holds true that*

$$p_\Phi^{\mathsf{Copeland}} = |\Phi| \cdot (2 \cdot n_{\mathsf{cand}} - |\Phi| - 1).$$

Proof. Let Φ be a dominating set. Consider a duel between the candidates $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}} \in C$:

1. None of the candidates is part of the dominating set, i.e., $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}} \in C \setminus \Phi$. Such duels do not affect the Copeland points of Φ.

2. Both are part of the dominating set, i.e., $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}} \in \Phi$. Then, this duel contributes two points to $p_\Phi^{\mathsf{Copeland}}$: Either one candidate received more votes than the other, then this candidate receives two points, and the other none, of both candidates received the same number of votes, them both candidates receive one point each.

3. Exactly one of them is part of the dominating set, i.e., $c_i^{\mathsf{cand}} \in \Phi, c_j^{\mathsf{cand}} \in C \setminus \Phi$. Since Φ is a dominating set, we have that $n_{\mathsf{votes}}^i > n_{\mathsf{votes}}^j$ and thus duel contributes two points to $p_\Phi^{\mathsf{Copeland}}$.

There are $\frac{|\Phi| \cdot (|\Phi|-1)}{2}$ duels of the second case and $|\Phi| \cdot (n_{\mathsf{cand}} - |\Phi|)$ duels of the third case. thus

$$\begin{aligned} p_\Phi^{\mathsf{Copeland}} &= 2 \cdot \frac{|\Phi| \cdot (|\Phi| - 1)}{2} + 2 \cdot |\Phi| \cdot (n_{\mathsf{cand}} - |\Phi|) \\ &= |\Phi| \cdot (|\Phi| - 1) + 2 \cdot |\Phi| \cdot (n_{\mathsf{cand}} - |\Phi|) \\ &= |\Phi| \cdot (|\Phi| - 1 + 2 \cdot n_{\mathsf{cand}} - 2 \cdot |\Phi|)) \\ &= |\Phi| \cdot (2 \cdot n_{\mathsf{cand}} - |\Phi| - 1)) \end{aligned}$$

For the other direction, let $S \subseteq C$ such that $p_S^{\mathsf{Copeland}} = |S| \cdot (2 \cdot n_{\mathsf{cand}} - |S| - 1)$. Again, we consider duels between all possible candidates $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}} \in C$, as done above:

1. None of the candidates is part of the set, i.e., $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}} \in C \setminus S$. Such duels do not affect the Copeland points of S.

2. Both are part of the set, i.e., $c_i^{\mathsf{cand}}, c_j^{\mathsf{cand}} \in S$. Then, this duel contributes two points to p_S^{Copeland}, with the same argumentation as above.

3. Exactly one of them is part of the set, i.e., $c_i^{\mathsf{cand}} \in S, c_j^{\mathsf{cand}} \in C \setminus S$. Let p^* be the sum of all points of this case.

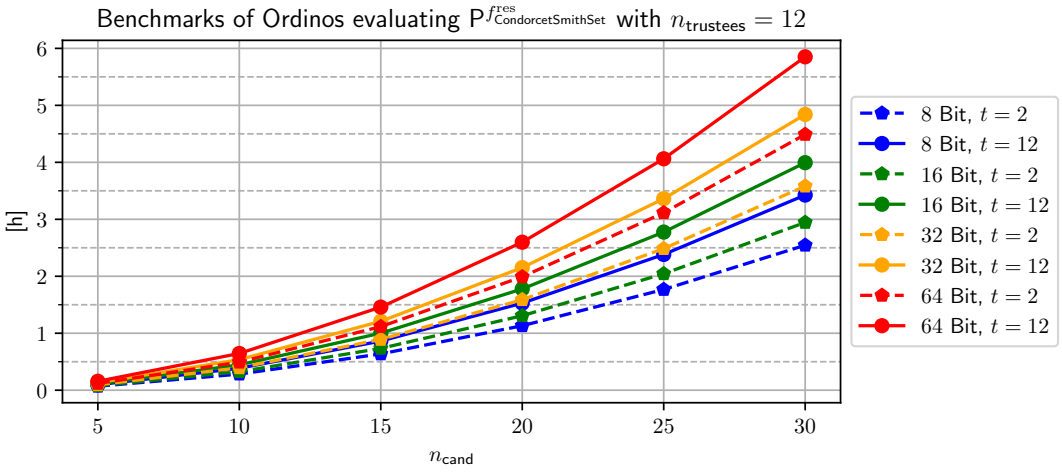


Figure 3.18.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSet}}}$ with $n_{\text{trustees}} = 12$.

Again, there are $\frac{|S|\cdot(|S|-1)}{2}$ duels of the second case. Since there are no other cases left, it must hold that $p^* = p_S^{\text{Copeland}} - 2 \cdot \frac{|C|\cdot(|C|-1)}{2}$.

$$
\begin{aligned}
p^* &= p_S^{\text{Copeland}} - 2 \cdot \frac{|S| \cdot (|S|-1)}{2} \\
&= |S| \cdot (2 \cdot n_{\text{cand}} - |S| - 1) - 2 \cdot \frac{|S| \cdot (|S|-1)}{2} \\
&= |S| \cdot (2 \cdot n_{\text{cand}} - |S| - 1 - |S| + 1) \\
&= |S| \cdot (2 \cdot n_{\text{cand}} - 2 \cdot |S|) \\
&= 2 \cdot |S| \cdot (n_{\text{cand}} - |S|)
\end{aligned}
$$

Since there are $|S| \cdot (n_{\text{cand}} - |S|)$ of this case, on average, each such duel must contribute to p^* with $\frac{2\cdot|S|\cdot(n_{\text{cand}}-|S|)}{|S|\cdot(n_{\text{cand}}-|S|)} = 2$ points. Since there is no possibility of obtaining more than two points with one duel, each such duel must contribute precisely two points, and thus, each candidate in S wins every duel against all candidates not in S. Therefore, S is a dominating set.

□

The secure tally-hiding MPC realization $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSet}}}$ for the Smith set makes use of Lemma 3.1 works as follows. Starting with the highest Condorcet points possible and iterating down to one, it adds all candidates that received this number of Copeland points to the current set. If the current set satisfies the requirement of Lemma 3.1, this set is the smallest possible dominating set and is, therefore, the Smith set. We present the complete protocol in Algorithm 3.16.

Figure 3.18 presents the benchmarks of $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSet}}}$, which scale quadratic in n_{cand}.

Algorithm 3.16: $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetSmithSet}}}$

Input: $E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})$

$E^{\mathsf{Matrix}}_{\mathsf{pk}}(D), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}}), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gt}}) = \mathsf{P}^{\mathsf{CondorcetHelper}}(E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}}))$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(b) = \text{CreateEncVector}(0, n_{\mathsf{cand}})$ ▷ Initialize result vector

$E_{\mathsf{pk}}(n) = E_{\mathsf{pk}}(0)$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(p^{\mathsf{Copeland}}) = (E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gt}})_i + E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}})_i)^{n_{\mathsf{cand}}}_{i=1}$ ▷ Compute Copeland points

$E_{\mathsf{pk}}(p^{\Phi}) = E_{\mathsf{pk}}(0)$

$E_{\mathsf{pk}}(f) = E_{\mathsf{pk}}(0)$ ▷ f saves whether the Smith set is found

for $t = 2 \cdot (n_{\mathsf{cand}} - 1)$ ***downto*** 1 **do** ▷ Iterate over possible Copeland points

  for $i \in \{1, \ldots, n_{\mathsf{cand}}\}$ **do**

    $E_{\mathsf{pk}}(e) = f_{\mathsf{eq}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(p^{\mathsf{Copeland}})_i, E_{\mathsf{pk}}(t))$ ▷ Check Copeland points of candidate

    $E^{\mathsf{Vector}}_{\mathsf{pk}}(b)_i = E^{\mathsf{Vector}}_{\mathsf{pk}}(b)_i \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(f)) \odot E_{\mathsf{pk}}(e)$

    $E_{\mathsf{pk}}(n) = E_{\mathsf{pk}}(n) \oplus E^{\mathsf{Vector}}_{\mathsf{pk}}(b)_i$ ▷ Update size of dominating set

    $E_{\mathsf{pk}}(p^{\Phi}) = E_{\mathsf{pk}}(p^{\Phi}) \oplus (E_{\mathsf{pk}}(t) \odot E^{\mathsf{Vector}}_{\mathsf{pk}}(b)_i)i$ ▷ Update Copeland points

  $E_{\mathsf{pk}}(t_{\text{tmp}}) = E_{\mathsf{pk}}(n) \odot (E_{\mathsf{pk}}(2) \odot E_{\mathsf{pk}}(n_{\mathsf{cand}}) \ominus E_{\mathsf{pk}}(n) \ominus E_{\mathsf{pk}}(1))$

  $E_{\mathsf{pk}}(f) = E_{\mathsf{pk}}(f) \oplus f_{\mathsf{eq}}(E_{\mathsf{pk}}(p^{\Phi}), E_{\mathsf{pk}}(t_{\text{tmp}}))$ ▷ Check if Smith set is found

return $(f_{\mathsf{dec}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(b)_i))^{n_{\mathsf{cand}}}_{i=1}$

3.2.3.15. Minimax

The method $f^{\text{res}}_{\mathsf{CondorcetMiniMax},\kappa}$ is parameterized by some metric κ, and the idea of this method is only to consider the *worst* duel of each candidate and then output all candidates that have the *best* of these worst duels. For this purpose, one needs to define some metric κ to assign scores $\theta \in \{0, \ldots, n_{\mathsf{votes}}\}$ to the duels, i.e., the *worst* (or *best*) duel is the one with the lowest (or highest) score. The selection of this metric κ defines a concrete instantiation of the minimax method. The tally-hiding minimax MPC protocol $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetMiniMax},\kappa}}$ for a generic metric κ is presented in Algorithm 3.17. This protocol computes for each candidate the score of the *worst* duel (according to the given metric) and then finds the candidate with the *best worst* score. We have implemented the minimax method with two metrics for Ordinos, namely *margins* and *winning votes*.

MiniMax Margins. The metric *margins*, denoted by $f_{\mathsf{MarginMetric}}$, sets the score of candidate c^{cand}_i's duel versus candidate c^{cand}_j to the difference between the number of duels won by c^{cand}_i versus c^{cand}_j minus the number of duels lost by c^{cand}_i versus c^{cand}_j. In order to scale the score in the correct range, we add the number of votes to this score. The tally-hiding realization is presented in Algorithm 3.18.

MiniMax Winning Votes. The metric *winning votes*, denoted by $f_{\mathsf{WinningVotesMetric}}$, sets the score of c^{cand}_i's duel versus c^{cand}_j to 0 if c^{cand}_i wins more duels versus c^{cand}_j than c^{cand}_i loses. Otherwise, it is given by the negative number of won duels of c^{cand}_j versus c^{cand}_i. We scale this score in the correct range by adding the number of votes. The tally-hiding realization is presented in Algorithm 3.19.

Algorithm 3.17: $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetMiniMax},\kappa}}$

Input: $E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})$

$E^{\mathsf{Matrix}}_{\mathsf{pk}}(D), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gteq}}), E^{\mathsf{Vector}}_{\mathsf{pk}}(\mathsf{d}^{\mathsf{gt}}) = \mathsf{P}^{\mathsf{CondorcetHelper}}(E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}}))$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(l^\theta) = (E_{\mathsf{pk}}(n_{\mathsf{votes}}))_{i=1}^{n_{\mathsf{cand}}}$ ▷ Initialize result vector

for $i \in \{1, \dots, n_{\mathsf{cand}}\}$ **do**

  for $j \in \{1, \dots, n_{\mathsf{cand}}\} \setminus \{i\}$ **do**

    $E_{\mathsf{pk}}(\theta) = \kappa(E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})_{ij}, E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})_{ji}, E_{\mathsf{pk}}(n_{\mathsf{votes}}))$ ▷ Compute score

    $E_{\mathsf{pk}}(b) = E^{\mathsf{Matrix}}_{\mathsf{pk}}(D)_{ji}$ ▷ Indicator for update

    $E_{\mathsf{pk}}(\theta) = E_{\mathsf{pk}}(b) \odot E_{\mathsf{pk}}(\theta) \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(b)) \odot E_{\mathsf{pk}}(n_{\mathsf{votes}})$

    $E_{\mathsf{pk}}(g') = f_{\mathsf{gt}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(l^\theta)_i, E_{\mathsf{pk}}(\theta))$

    $E^{\mathsf{Vector}}_{\mathsf{pk}}(l^\theta)_i = E_{\mathsf{pk}}(g') \odot E_{\mathsf{pk}}(\theta) \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g')) \odot E^{\mathsf{Vector}}_{\mathsf{pk}}(l^\theta)_i$ ▷ Update score

return $\mathsf{P}^{f^{\text{res}}_{\mathsf{Plurality}}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(l^\theta))$

Algorithm 3.18: $\mathsf{P}^{f_{\mathsf{MarginMetric}}}$

Input: $E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})_{ij}, E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})_{ji}, E_{\mathsf{pk}}(n_{\mathsf{votes}})$

return $E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})_{ij} \ominus E^{\mathsf{Matrix}}_{\mathsf{pk}}(\mathsf{DM}^{\mathsf{agg}})_{ji} \oplus E_{\mathsf{pk}}(n_{\mathsf{votes}})$

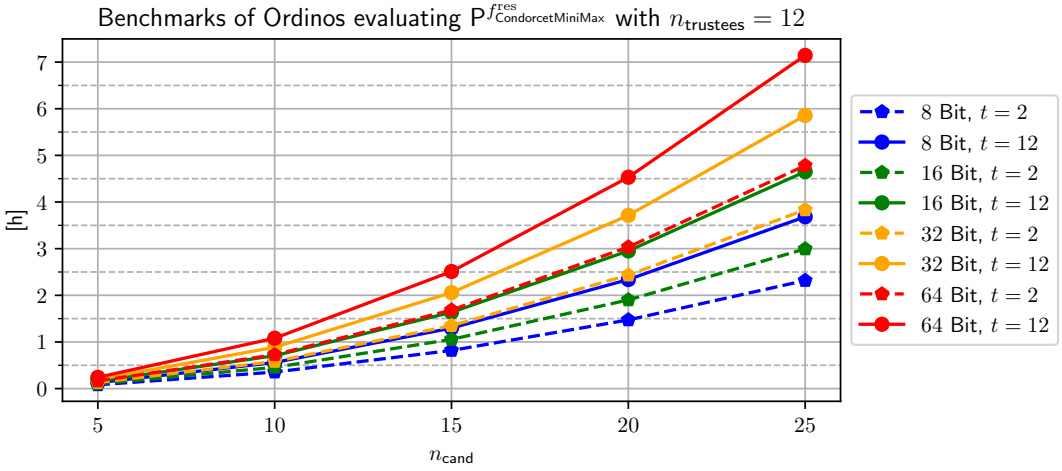


Figure 3.19.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetMiniMax},\kappa}}$ for $\kappa = f_{\mathsf{MarginMetric}}$ and $\kappa = f_{\mathsf{WinningVotesMetric}}$ with $n_{\mathsf{trustees}} = 12$.

In Figure 3.19, we present the benchmarks of $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetMiniMax},\kappa}}$ for both metrics. Since both methods only employ additions and subtractions, they provide equal runtime.

3.2.3.16. Schulze

The Schulze method $f^{\text{res}}_{\mathsf{CondorcetSchulze}}$ is a slightly more complicated, but commonly used Condorcet method (see, e.g., [Sch18]). In this method, we also consider a score function for the duels, which can be the same as in the Copeland method. The candidates and the duels between them are considered as a complete directed weighted graph Γ, where the nodes of Γ represent the

Algorithm 3.19: $\mathsf{P}^{f_{\mathsf{WinningVotesMetric}}}$

Input: $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ij}, E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ji}, E_{\mathsf{pk}}(n_{\mathsf{votes}})$

return $E_{\mathsf{pk}}(n_{\mathsf{votes}}) \ominus E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ji}$

Algorithm 3.20: $\mathsf{P}^{f^{\mathrm{res}}_{\mathsf{CondorcetSchulze}}}$

Input: $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})$

$E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma) = \mathrm{CreateEncMatrix}(0, n_{\mathsf{cand}}, n_{\mathsf{cand}})$ ▷ Directed weighted graph

for $i \in \{1, \ldots, n_{\mathsf{cand}}\}$ **do** ▷ Initalize directed weighted graph

  for $j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\}$ **do**

    $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma)_{ij} = E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ij} \ominus E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ji}$

for $i \in \{1, \ldots, n_{\mathsf{cand}}\}, j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\}, k \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i, j\}$ **do** ▷ Compute directed weighted graph

  $E_{\mathsf{pk}}(m) = \mathsf{P}^{\mathrm{MinVal}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma_{i,k}), E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma_{k,j}))$

  $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma)_{ij} = \mathsf{P}^{\mathrm{MaxVal}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\mathsf{DM}^{\mathsf{agg}})_{ij}, E_{\mathsf{pk}}(m))$

$E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Omega) = \mathrm{CreateEncMatrix}(0, n_{\mathsf{cand}}, n_{\mathsf{cand}})$ ▷ Path value matrix

for $i \in \{1, \ldots, n_{\mathsf{cand}}\}, j \in \{1, \ldots, i-1\}$ **do** ▷ Compute path value matrix

  $E_{\mathsf{pk}}(g) = f_{\mathsf{gt}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma)_{ij}, E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma)_{ji})$

  $E_{\mathsf{pk}}(e) = f_{\mathsf{eq}}(E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma)_{ij}, E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Gamma)_{ji})$

  $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Omega)_{ij} = E_{\mathsf{pk}}(g)$

  $E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Omega)_{ji} = E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(g) \oplus E_{\mathsf{pk}}(e)$

$E_{\mathsf{pk}}^{\mathsf{Vector}}(b) = \mathrm{CreateEncVector}(0, n_{\mathsf{cand}})$ ▷ Result vector

for $i \in \{1, \ldots, n_{\mathsf{cand}}\}$ **do**

  $E_{\mathsf{pk}}(w) = E_{\mathsf{pk}}(0)$

  for $j \in \{1, \ldots, n_{\mathsf{cand}}\} \setminus \{i\}$ **do**

    $E_{\mathsf{pk}}(w) = E_{\mathsf{pk}}(w) \oplus E_{\mathsf{pk}}^{\mathsf{Matrix}}(\Omega)_{ij}$

  $E_{\mathsf{pk}}^{\mathsf{Vector}}(b)_i = f_{\mathsf{eq}}(E_{\mathsf{pk}}(w), E_{\mathsf{pk}}(n_{\mathsf{cand}}) \ominus E_{\mathsf{pk}}(1))$

return $(f_{\mathsf{dec}}(E_{\mathsf{pk}}^{\mathsf{Vector}}(b)_i))_{i=1}^{n_{\mathsf{cand}}}$

candidates and an arrow $c_i^{\mathsf{cand}} \to c_j^{\mathsf{cand}}$ is weighted with the score of c_i^{cand}'s duel versus c_j^{cand}. Now, for any path p in Γ, we define the *value* of p as the lowest weight among the arrows involved in p. We then consider the *path value matrix* Ω, an $(n_{\mathsf{cand}} \times n_{\mathsf{cand}})$-matrix with entry Ω_{ij} being the highest path value among paths from c_i^{cand} to c_j^{cand}. The Schulze method then outputs all candidates c_i^{cand} such that $\Omega_{ij} \geq \Omega_{ji}$ for each $j \in \{1, \ldots, n_{\mathsf{cand}}\}$. We note that the Schulze method always outputs some candidate(s). With the metrics described in the Copeland method, this candidate is the unique Condorcet winner, if existent. We present the secure tally-hiding MPC protocol for the Schulze evaluation in Algorithm 3.20.

Figure 3.20 presents the benchmarks of $\mathsf{P}^{f^{\mathrm{res}}_{\mathsf{CondorcetSchulze}}}$, which are cubic in n_{cand}.

3.2.3.17. Instant-Runoff Voting (IRV)

Our Ordinos instantiation supports the instant-runoff election result functions $f^{\mathrm{res}}_{\mathsf{IRVLotComplete}}$, $f^{\mathrm{res}}_{\mathsf{IRVLotPartial}}$, $f^{\mathrm{res}}_{\mathsf{IRVNSWCompelte}}$, and $f^{\mathrm{res}}_{\mathsf{IRVNSWPartial}}$. We interpret a ballot as a ranking permutation

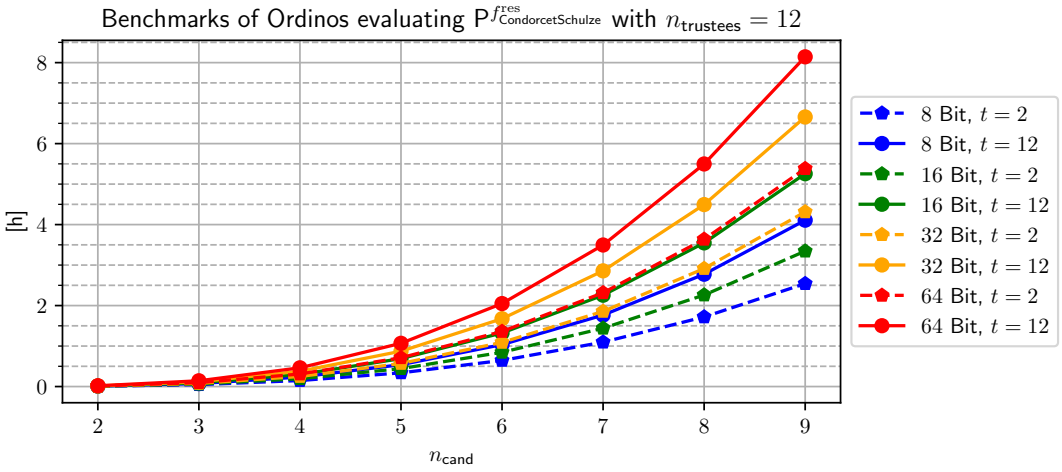


Figure 3.20.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSchulze}}}$ with $n_{\text{trustees}} = 12$.

(see Section 2.6). As pointed out in Section 2.6, the number of possible choices grows factorial, respective exponential, in the number of candidates. For example, as presented in Table 2.1, for partial ranking, there are $1,957$ possible choices for 6 candidates, while for 7 candidates, there are already $13,700$ possible choices.

We present the tally-hiding realization $\mathsf{P}^{f^{\text{res}}_{\text{IRV}}}$ in Algorithm 3.21. We can instantiate this algorithm to support any instant-runoff method mentioned above. The difference between the two methods is the tie-breaking mechanism. Our tally-hiding instantiation uses an algorithm GetLastTieBreaking, which we can instantiate to support the concrete instant-runoff method we want to evaluate. Furthermore, our tally-hiding protocol works for both rankings, complete and partial. The difference is which (choice) components of the aggregated tally the algorithm will consider to compute the points per candidate in each round.

We use an indicator vector $(X_i)_{i=1}^{n_{\text{cand}}}$ such that $X_i = 0$ if c_i^{cand} is not eliminated and $X_i = 1$ if c_i^{cand} is eliminated. We compute which candidate to give the corresponding votes for each ranking. We search for the first not-eliminated preference for each ranking. For this, we obtain an encrypted indicator c_{win} that contains the candidate's index. We then check for each candidate that occurs in the ranking whether the indices match and add the votes accordingly. In the second phase, we compute the indicator vector using a tie-breaking mechanism, which states which candidates received the least votes. We then update X accordingly.

On the surface, the algorithm's complexity to evaluate IRV appears to scale quadratic in the number of candidates: The algorithm iterates over $n_{\text{cand}} - 1$ rounds, and in each round, we search for and eliminate the candidate that received the least number of votes. However, the main factor of this algorithm's complexity lies in computing the votes per candidate in each round. To collect all candidate votes, we must iterate through all (choice) components, leading to a complexity of $n_{\text{cand}} \cdot n_{\text{components}}$ in each round. Since the number of (choice) components scales exponentially in the number of candidates, this step becomes very expensive, even for small numbers of candidates. Luckily, the algorithm does not perform expensive comparison

Algorithm 3.21: $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{IRV}}}$

Input: $E^{\mathsf{Vector}}_{\mathsf{pk}}(c^{\mathsf{choice}}_{\mathsf{agg}})$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(X) = \mathrm{CreateEncVector}(0, n_{\mathsf{cand}})$ ▷ Encrypted indicator bits.

for $i \in \{1, \ldots, n_{\mathsf{cand}} - 1\}$ **do** ▷ Perform $n_{\mathsf{cand}} - 1$ elimination rounds

  $E^{\mathsf{Vector}}_{\mathsf{pk}}(v^{\mathsf{round},i}) = \mathrm{CreateEncVector}(0, n_{\mathsf{cand}})$ ▷ Votes received in this round.

  for *(ordered) i-tuple* r_i *with entries in* $\{1, \ldots, n_{\mathsf{cand}}\}$ **do** ▷ Go over ranking prefixes

    $E_{\mathsf{pk}}(c_{\mathsf{win}}) = E_{\mathsf{pk}}(0)$ ▷ c_{win} will be the winner of the prefix

    $E_{\mathsf{pk}}(d) = E_{\mathsf{pk}}(0)$ ▷ The bit d indicates whether we already found the winner

    for c *in* r_i **do** ▷ Find winner in prefix

      $E_{\mathsf{pk}}(c_{\mathsf{win}}) = E_{\mathsf{pk}}(d) \odot E_{\mathsf{pk}}(c_{\mathsf{win}}) \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(d)) \odot E_{\mathsf{pk}}(c)$

      $E_{\mathsf{pk}}(d) = E_{\mathsf{pk}}(d) \oplus (E_{\mathsf{pk}}(1) \ominus E_{\mathsf{pk}}(d)) \odot (E_{\mathsf{pk}}(1) \ominus E^{\mathsf{Vector}}_{\mathsf{pk}}(X)_c)$

    for c *in* r_i **do** ▷ Add points from ballots for current prefix to the winner

      $E_{\mathsf{pk}}(w) = f_{\mathsf{eq}}(E_{\mathsf{pk}}(c), E_{\mathsf{pk}}(c_{\mathsf{win}}))$

      for $j \in \{1, \ldots, n_{\mathsf{components}}\}$ *s.t.* j *represents a ranking where the top* i *candidates are* r_i **do**

        $E^{\mathsf{Vector}}_{\mathsf{pk}}(v^{\mathsf{round},i})_c = E^{\mathsf{Vector}}_{\mathsf{pk}}(v^{\mathsf{round},i})_c \oplus E_{\mathsf{pk}}(w) \odot E^{\mathsf{Vector}}_{\mathsf{pk}}(c^{\mathsf{choice}}_{\mathsf{agg}})_j$

  $E^{\mathsf{Vector}}_{\mathsf{pk}}(l) = \mathrm{GetLastTieBreaking}(E^{\mathsf{Vector}}_{\mathsf{pk}}(X))$

  for $c \in \{1, \ldots, n_{\mathsf{cand}}\}$ **do** ▷ Update/add one eliminated candidate

    $E^{\mathsf{Vector}}_{\mathsf{pk}}(X)_c = E^{\mathsf{Vector}}_{\mathsf{pk}}(X)_c \oplus (E_{\mathsf{pk}}(1) \ominus E^{\mathsf{Vector}}_{\mathsf{pk}}(X)_c) \odot E^{\mathsf{Vector}}_{\mathsf{pk}}(l)_c$

return $f_{\mathsf{dec}}((E^{\mathsf{Vector}}_{\mathsf{pk}}(X)_c)_{c=1}^{n_{\mathsf{cand}}})$

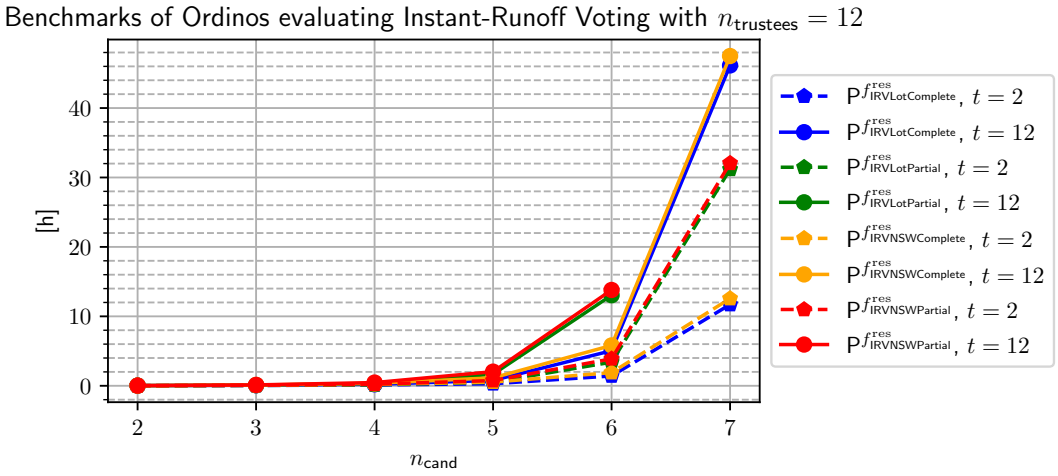


Figure 3.21.: Benchmarks of Ordinos evaluating IRV with $n_{\mathsf{trustees}} = 12$.

operations for each (choice) component but only multiplications. Interestingly, this is the only present multi-party computation protocol dominated by multiplications.

We provide benchmarks of the algorithm for all instant-runoff variants in Figure 3.21. The difference in multiplication runtimes for various thresholds is tiny, but applying many multiplications results in vastly different runtimes for various thresholds. Additionally, the tie-breaking mechanism scales in the number of candidates rather than the number of choice components, making the difference in runtime for different tie-breaking mechanisms relatively negligible

compared to the overall runtime of the algorithm. We note that the benchmarks for the following numbers of candidates did not finish in under five days.

3.2.3.18. Seat Allocation Methods

A crucial type of election in many countries worldwide is the parliamentary election, which involves choosing candidates based on party affiliation. These complex elections involve millions of voters, dozens of parties, and hundreds of electoral constituencies. Some voters have multiple votes to distribute among parties and individual candidates. Sophisticated algorithms compute election results by assigning seats to individual candidates. An essential component of such algorithms is a seat allocation method, which utilizes the number of available seats and the total number of votes for each party to compute the seats assigned to each party.

Real-world seat-allocation methods run multiple times on different inputs to compute the election results of parliamentary elections. For example, to evaluate the final seat distribution of the German Bundestag election in 2021, the Sainte-Laguë method is run 23 times. Hence, optimizing MPC components for seat-allocation methods is crucial for an efficient tally-hiding voting system.

We instantiated the Ordinos framework for two seat allocation methods, and we will present the tally-hiding realizations next.

3.2.3.19. Hare-Niemeyer Method

The MPC protocol $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{HareNiemeyer}}}$ computes the function $f^{\mathsf{res}}_{\mathsf{HareNiemeyer}}$ in a tally-hiding way. We present this protocol in Algorithm 3.22.

In the first phase, the algorithm obtains the division result rounded down. That is, for each c_i^{party}, we obtain an (encrypted) $s_i = \lfloor \frac{n^i_{\mathsf{votes}} \cdot n_{\mathsf{seats}}}{v_{\mathsf{total}}} \rfloor$. This value indicates how many seats each party receives without considering the remaining seats. Next, we compute how many remainder seats n_r we must distribute among the parties. We compute each party's remainder d_i. We can find the highest n_r values among the d_i's. Recall that the algorithm for this does not need to know the threshold in plain.

We present the benchmarks of $\mathsf{P}^{f^{\mathsf{res}}_{\mathsf{HareNiemeyer}}}$ in Figure 3.22.

3.2.3.20. Sainte-Laguë Method

We want to construct a tally-hiding MPC component that takes as inputs $E_{\mathsf{pk}}(n^j_{\mathsf{votes}})$ for each party as well as publicly known values n_{parties} and n_{seats} and computes the encrypted Sainte-Laguë seat distribution $E_{\mathsf{pk}}(n^j_{\mathsf{seats}})$. As an initial insight, we observe that basing the MPC protocol on the suitable-denominator algorithm is generally very inefficient: This algorithm has to iterate over several potential denominators d until it finds a suitable one. Since the number of iterations required to find d reveals non-trivial information about the secret inputs, the MPC protocol must run with a fixed number of iterations m, which we must choose large enough to ensure

Algorithm 3.22: $\mathsf{P}^{f^{\text{res}}_{\text{HareNiemeyer}}}$

Input: $E_{\mathsf{pk}}(n^1_{\text{votes}}), \ldots, E_{\mathsf{pk}}(n^{n_{\text{parties}}}_{\text{votes}}), n_{\text{seats}}, v_{\text{total}}$

1 $E^{\text{Vector}}_{\mathsf{pk}}(s) = \text{CreateEncVector}(0, n_{\text{parties}})$

2 **for** $i \in \{1, \ldots, n_{\text{parties}}\}$ **do** ▷ Division per party

3 $\quad E_{\mathsf{pk}}(t_{\text{tmp}}) = E_{\mathsf{pk}}(n^i_{\text{votes}}) \odot E_{\mathsf{pk}}(n_{\text{seats}})$

4 $\quad E^{\text{Vector}}_{\mathsf{pk}}(s)_i = \mathsf{P}^{\text{FloorDiv}}(E_{\mathsf{pk}}(t_{\text{tmp}}), E_{\mathsf{pk}}(v_{\text{total}}), n_{\text{seats}})$

5 $E_{\mathsf{pk}}(n_r) = E_{\mathsf{pk}}(n_{\text{seats}}) \ominus (\bigoplus_{i=1}^{n_{\text{parties}}} E^{\text{Vector}}_{\mathsf{pk}}(s)_i)$ ▷ Number of remaining seats

6 $E^{\text{Vector}}_{\mathsf{pk}}(d) = \text{CreateEncVector}(0, n_{\text{parties}})$

7 **for** $i \in \{1, \ldots, n_{\text{parties}}\}$ **do** ▷ Compute remaining dividers

8 $\quad E^{\text{Vector}}_{\mathsf{pk}}(d)_i = E_{\mathsf{pk}}(n^i_{\text{votes}}) \odot E_{\mathsf{pk}}(n_{\text{seats}}) \ominus v_{\text{total}} \odot E^{\text{Vector}}_{\mathsf{pk}}(s)_i$

9 $E^{\text{Vector}}_{\mathsf{pk}}(d^{\text{best}}) = \mathsf{P}^{f^{\text{res}}_{\text{Best}, E_{\mathsf{pk}}(n_r)}}(E^{\text{Vector}}_{\mathsf{pk}}(d))$ ▷ Find dividers of maximum value

10 **for** $i \in \{1, \ldots, n_{\text{parties}}\}$ **do**

11 $\quad E^{\text{Vector}}_{\mathsf{pk}}(s)_i = E^{\text{Vector}}_{\mathsf{pk}}(s)_i \oplus E^{\text{Vector}}_{\mathsf{pk}}(d^{\text{best}})_i$

12 **return** $f_{\mathsf{dec}}(E^{\text{Vector}}_{\mathsf{pk}}(s))$

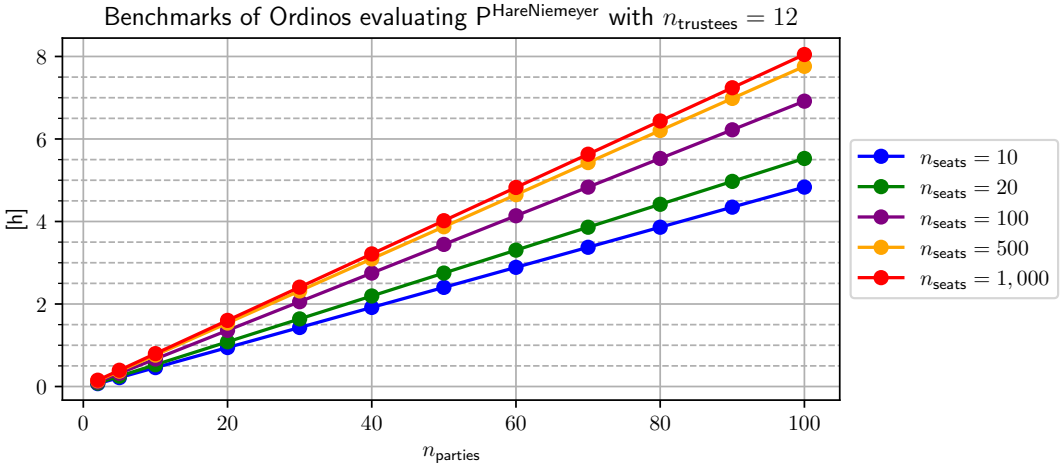


Figure 3.22.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{HareNiemeyer}}}$ with $n_{\text{trustees}} = 12$ and 64 bit values.

that we find a suitable divisor d for all possible input sequences. This worst-case approximation introduces extensive additional overhead.

We have therefore constructed a fundamental tally-hiding MPC realization $\mathsf{P}^{\text{SLQBasic}}$ of the Sainte-Laguë method following the highest-quotients approach: each party j is assigned its current quotient q_{current} (see the description of the highest-quotients algorithm) and seats n^j_{seats} thus far. Algorithm 3.23 shows this excerpt of a single iteration step. Note that this $\mathsf{P}^{\text{SLQBasic}}$ uses the fast $\mathsf{P}^{\text{MaxFractionLastIdx}}$ protocol in all iterations and, hence, breaks ties via a fixed mechanism that always assigns the seat to the party with the highest index j.

Support for breaking ties by lot. Many elections use more involved tie-breaking algorithms than the default one implemented by $\mathsf{P}^{\text{SLQBasic}}$. For example, for many parliamentary elections, e.g., elections in Indonesia, Sweden, and Germany, a new lot is drawn to resolve the tie whenever

Algorithm 3.23: $\mathsf{P}^{\mathsf{AddSeatBasic}}$

Input: $(E^{\mathsf{Frac}}_{\mathsf{pk}}(q^j_{\text{current}}))_{j=1}^{n_{\mathsf{parties}}}$, $(E^{\mathsf{Vector}}_{\mathsf{pk}}(n_{\mathsf{seats}})_j)_{j=1}^{n_{\mathsf{parties}}}$

$E^{\mathsf{Vector}}_{\mathsf{pk}}(t_{\text{tmp}}) = \mathsf{P}^{\text{MaxFractionLastIdx}}((E^{\mathsf{Frac}}_{\mathsf{pk}}(q^j_{\text{current}}))_{j=1}^{n_{\mathsf{parties}}})$

for $i \in \{1, \ldots, n_{\mathsf{parties}}\}$ **do**

  $E_{\mathsf{pk}}(d) = E^{\mathsf{FracVector}}_{\mathsf{pk}}(q_{\text{current}})_j.\mathsf{den} \oplus 2 \odot E^{\mathsf{Vector}}_{\mathsf{pk}}(t_{\text{tmp}})_j$

  $E^{\mathsf{Vector}}_{\mathsf{pk}}(q_{\text{current}})_j = \text{CreateEncFraction}(E^{\mathsf{FracVector}}_{\mathsf{pk}}(q_{\text{current}})_j.\mathsf{num}, E_{\mathsf{pk}}(d))$ ▷ Update q

  $E^{\mathsf{Vector}}_{\mathsf{pk}}(n_{\mathsf{seats}})_j = E^{\mathsf{Vector}}_{\mathsf{pk}}(n_{\mathsf{seats}})_j \oplus E^{\mathsf{Vector}}_{\mathsf{pk}}(t_{\text{tmp}})_j$ ▷ Update seats (n_{seats})

return $f_{\mathsf{dec}}(E^{\mathsf{Vector}}_{\mathsf{pk}}(s))$

there is a tie between several parties for a seat. A more general tally-hiding MPC implementation $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ for this election has to support this tie-breaking mechanism and keep secret whether it has drawn any lots. In particular, to build such a $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ we first need to extend and modify the iteration step AddSeatBasic shown in Algorithm 3.23, obtaining a new subroutine AddSeatTieBreaking which takes as additional input an encrypted ranking of parties $r = (E_{\mathsf{pk}}(r_0), \ldots, E_{\mathsf{pk}}(r_{n_{\mathsf{parties}}-1}))$ where r is a uniformly chosen permutation of $0, \ldots, n_{\mathsf{parties}} - 1$, and then resolves ties based on that ranking.

We construct AddSeatTieBreaking by making use of $\mathsf{P}^{\text{MaxFractionByRank}}$ as presented in Algorithm 3.8. That is, we replace the call to $\mathsf{P}^{\text{MaxFractionLastIdx}}$ of AddSeatBasic by our algorithm $\mathsf{P}^{\text{MaxFractionByRank}}$, which takes as additional input the ranking r. Based on this AddSeatTieBreaking, we have constructed two versions of a $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ MPC component which implements Sainte-Laguë. In each iteration, these MPC components first compute an encrypted ranking r that encodes the tie-breaking result and then use AddSeatTieBreaking with that r. We introduce two main optimizations in both cases: First, for tie-breaking by lot, we run a distributed randomness generation protocol [LT13] for each iteration to compute r based on the results. Since this step is input/vote independent, it can be pre-computed before the election. Secondly, observe that if a tie occurs between m parties in one iteration of the quotient approach while we have at least m seats to distribute, all parties in the tie will obtain a seat in the subsequent m iterations, i.e., it does not matter how we break this tie. Hence, only ties during the last $n_{\mathsf{parties}} - 1$ iterations need to be handled by AddSeatTieBreaking, while otherwise we use the faster AddSeatBasic algorithm.

Secret number of seats. We note that we can extend all of our MPC algorithms for the tally-hiding baseline quotient method (as well as for the new method presented subsequently) to run with a secret, i.e., encrypted, total number of seats n_{seats} to be assigned, as long as an upper limit of seats is known. We use this property for the tally-hiding voting system for the German elections, where the Sainte-Laguë method is used, among others, on the number of seats assigned to a specific constituency, which is an intermediate result that might have to remain secret depending on the desired tally-hiding property.

Sainte-Laguë based on floor division. While our multi-party computation protocols $\mathsf{P}^{\mathsf{SLQBasic}}$ and $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ based on the highest-quotients approach are already practical in terms

of efficiency, they always use n_{seats} iterations to assign all seats and thus do not scale overly well for elections allocating a high number of seats n_{seats}. To improve performance in such cases, we propose a new algorithm for computing the Sainte-Laguë method, called the *floor division method*. Our floor division method is different from the highest-quotient and the suitable-denominator methods. It allows us to construct an MPC component, called $\mathsf{P}^{\mathsf{SLQFloorDiv}}$, that requires n_{parties} instead of n_{seats} many iterations, and thus, is more efficient in the typical case that the number of seats exceeds the number of parties. In what follows, we present the floor division method and describe our MPC component $\mathsf{P}^{\mathsf{SLQFloorDiv}}$.

The concept of our novel method is as follows. The floor division method replaces initial iterations and seat assignments with under- and overestimating final seat allocation. We run only the final (at most n_{parties} many) iterations of the quotient method to add/remove a seat from the initial estimations until exactly n_{seats} seats are assigned. As we prove, we can efficiently determine the correct Sainte-Laguë distribution.

Concretely, we compute $m_j := \lfloor \frac{n^j_{\mathsf{votes}} \cdot n_{\mathsf{seats}}}{n_{\mathsf{votes}}} \rfloor$ for each c_j^{party}. We assign m_j seats to c_j^{party} for the underestimation case. Note that $s^{\mathsf{min}}_{\mathsf{initial}} := \sum_{j \in \{1,\ldots,n_{\mathsf{parties}}\}} m_j$ might be smaller than n_{seats}, but not smaller than $n_{\mathsf{seats}} - n_{\mathsf{parties}}$. Hence, to distribute exactly n_{seats} seats in total, we distribute the remaining $n_{\mathsf{seats}} - s^{\mathsf{min}}_{\mathsf{initial}}$ ($\leq n_{\mathsf{parties}}$) seats to the parties by executing the final iterations of the highest-quotients method (and the desired tie-breaking mechanism). We start with the intermediate quotients $m^j_{m_j} := \frac{n^j_{\mathsf{votes}}}{2m_j+1}$ instead of the initial m^j_0 for each c_j^{party}. For the overestimation case, we start by assigning $m_j + 1$ seats to c_j^{party}, which might assign at most n_{parties} additional seats beyond the desired total of n_{seats}. We use a reverse variant of the highest-quotients algorithm to remove those seats. For this purpose, we again initialize the quotients as $m^j_{m_j}$ and then, in each iteration step, determine the minimal current quotient and remove a seat from the corresponding party (using the desired tie-breaking mechanism). Next, we update the quotient of that party by reducing the denominator by 2. We could remove all $m_j + 1$ seats from a c_j^{party}. In that case, we ignore this party in the following iterations. This special case is non-trivial to implement in our $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ MPC component since we cannot reveal the values m_j or the quotients. We continue this iteration until only we distribute a total of n_{seats} seats.

Finally, to deduce which outcome is the correct Sainte-Laguë distribution, we evaluate the underestimation case an additional time to compute the next seat that would be assigned. If the corresponding quotient is less than all the initial quotients $m^j_{m_j}$ of the underestimation, then the underestimation computed the correct seat distribution. Otherwise, the correct seat distribution is computed based on the overestimation. We show the following result:

Lemma 3.2 (Correctness of $\mathsf{P}^{\mathsf{SLQFloorDiv}}$)**.** *The algorithm $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ as presented above is correct, i.e., always outputs the seat allocation according to the Sainte-Laguë method with the desired tie-breaking mechanism.*

To prove Lemma 3.2, we first make the connection between the values q_{current} occurring during the execution of $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ and the highest quotients from the generic description of

the Sainte-Laguë method explicit. By construction, all of the n_{seats} highest quotients must occur as a maximal value q_{current} at some iteration step during the execution of $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$. Suppose a value $q^j_{\text{current}} = m^j_i$ is not a maximal value in the first n_{seats} iteration steps of the execution of $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$. In that case, it does not belong to the highest quotients of the n_{seats}; hence, c^{party}_j does not receive the i-th seat.

Next, we will require the following lemma, where we follow Genssler [Gen84]:

Lemma 3.3 (Over- and Underestimation)**.** *For $j \in \{1, \ldots, n_{\mathsf{parties}}\}$ denote $m_j := \lfloor \frac{n^j_{\mathsf{votes}} \cdot n_{\mathsf{seats}}}{n_{\mathsf{votes}}} \rfloor$. If in a Sainte-Laguë distribution, a party i is assigned $m_i + 2$ seats, then* all *parties j receive at least m_j seats.*

Proof. We compare the quotients $m^j_{m_j}$ and $m^i_{m_i+2}$. In the case that $m_j = 0$, the statement is trivial. If $m_i = 0$ and $m_j = 1$, we have $n^j_{\mathsf{votes}} > n^i_{\mathsf{votes}}$, so party j must be assigned (at least) as many seats as party i. Thus, if party i is assigned $m_i + 2 = 2$ seats, then also party j is assigned (at least) $2 > 1 = m_j$ seats, and the statement holds. In all other cases, we know that $m_j + m_i - 1 > 0$ and can reason as follows:

$$\begin{aligned} m_j > -(m_i + 1) &\implies 2m_j m_i + 3m_j > 2m_j m_i - (m_i + 1) + 2m_j \\ &\implies m_j \cdot (2m_i + 3) > (m_i + 1) \cdot (2m_j - 1) \\ &\implies \frac{m_j}{2m_j - 1} > \frac{m_i + 1}{2m_i + 3} \quad (*). \end{aligned}$$

Now, using that $n^j_{\mathsf{votes}} \cdot \frac{n_{\mathsf{seats}}}{n_{\mathsf{votes}}} = m_j + a_j$ for some $a_j \in [0, 1)$ and the fact that $\frac{n_{\mathsf{seats}}}{n_{\mathsf{votes}}} > 0$, we can conclude from $(*)$ as follows:

$$(*) \implies \frac{m_j + a_j}{2m_j - 1} > \frac{m_i + a_i}{2m_i + 3} \implies \frac{n^j_{\mathsf{votes}}}{2m_j - 1} > \frac{n^i_{\mathsf{votes}}}{2_i + 3},$$

i.e., $m^j_{m_j} > m^i_{m_i+2}$ and hence if $m^i_{m_i+2}$ belongs to the n_{seats} highest quotients, then also all $m^j_{m_j+2}$ belong to the n_{seats} highest quotients. □

These preparations now allow for proving the correctness of $\mathsf{P}^{\mathsf{SLQFloorDiv}}$:

Proof of Lemma 3.2. We use that Lemma 3.3 allows us to distinguish two cases:

(i) There is a party i that in the seat allocation according to the Sainte-Laguë method receives at least $m_i + 2$ seats.

(ii) Each party j receives at most $m_j + 1$ seats in the seat allocation according to the Sainte-Laguë method.

Let us first consider case (i). In this case, each party j receives at least m_j seats by the previous lemma. Thus, we find that all quotients m^j_i for $i \leq m_j$ must belong to the n_{seats} highest quotients and hence occur as maximal values q^j_{current} during one of the first n_{seats} iteration steps

of $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$. Since the order in which seats are allocated in the Sainte-Laguë method does not affect the final result, one can therefore start by assigning m_j seats to each party and then allocate the remaining seats using $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ starting with the quotients $m^j_{m_j}$ that match the setting that each party j has been assigned m_j seats. This is exactly what the first subroutine in $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ does. Moreover, since in case (i) all of the values $m^j_{m_j}$ must belong to the n_{seats} largest quotients, we find that - if we follow the $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ algorithm - the maximal value q_{current} at the end of the computation of the first subroutine must be less than (in this comparison we take tie-breaking into account) all of the values $m^j_{m_j}$, since this value of q_{current} is the first one to not belong to the n_{seats} largest quotients. Hence, in case (i) the seat allocation output by $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ is the result from the first subroutine, which as we argued, is the correct seat allocation according to the Sainte-Laguë method.

Suppose we fall in case (ii). In that case, it can still happen that the maximal value q_{current} at the end of the computation of the first subroutine is less than or equal to all of the values $m^j_{m_j}$. Namely, this happens if all parties j receive m_j or $m_j + 1$ seats. In that case, we can again argue as above to see that $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ returns the correct seat allocation. However, if a party i receives less than m_i seats, then $m^i_{m_i}$ does *not* belong to the n_{seats} highest quotients and thus the check at the end of the first subroutine will fail. In this case, $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ will return the result obtained from the second subroutine. However, since we fall in case (ii), we know that each party j has received at most $m_j + 1$ seats, so if we start by assigning $m_j + 1$ seats to each party j, we know that no party has received too few seats and can then remove the surplus seats.

In this case, if we denote $s^{\mathsf{max}}_{\mathsf{initial}} := \sum_{j\in\{1,\ldots,n_{\mathsf{parties}}\}}(m_j + 1)$ and let M^{over} be the list of quotients m^j_j for $i \leq m_j$, then the list M from the generic description of the highest-quotients method (see Section 2.7) consists exactly of the n_{seats} largest entries in M^{over}. Thus, if we remove the $\beta = s^{\mathsf{max}}_{\mathsf{initial}} - n_{\mathsf{seats}}$ seats corresponding to the β smallest quotients (with tie-breaking) in M^{over}, then each party is assigned the correct number of seats according to the Sainte-Laguë method.

This is exactly what $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ does in the second subroutine. More precisely, by starting with the quotients $q^j_{\text{current}} = m^j_{m_j}$, the minimal value q^j_{current} is exactly the minimal value in M^{over}. Removing a seat from the corresponding party and updating the quotient by reducing its denominator by 2 then exactly corresponds to removing the minimal element from M^{over}. Repeating this process until a total of n_{seats} seats remain assigned corresponds exactly to removing the β seats corresponding to the β smallest quotients in M^{over} and hence the result of the second subroutine in this case is exactly the seat allocation according to the Sainte-Laguë method. Recall that we ignore quotients for parties j from which $m_j + 1$ seats have already been removed. □

Tally-Hiding MPC component. When constructing our tally-hiding MPC protocol for computing the Sainte-Laguë method, the primary concern is to ensure that the algorithm does not reveal the number of iterations required to add or subtract seats from the initial seat assignment. This is important because it could expose valuable information. To address this issue, we have

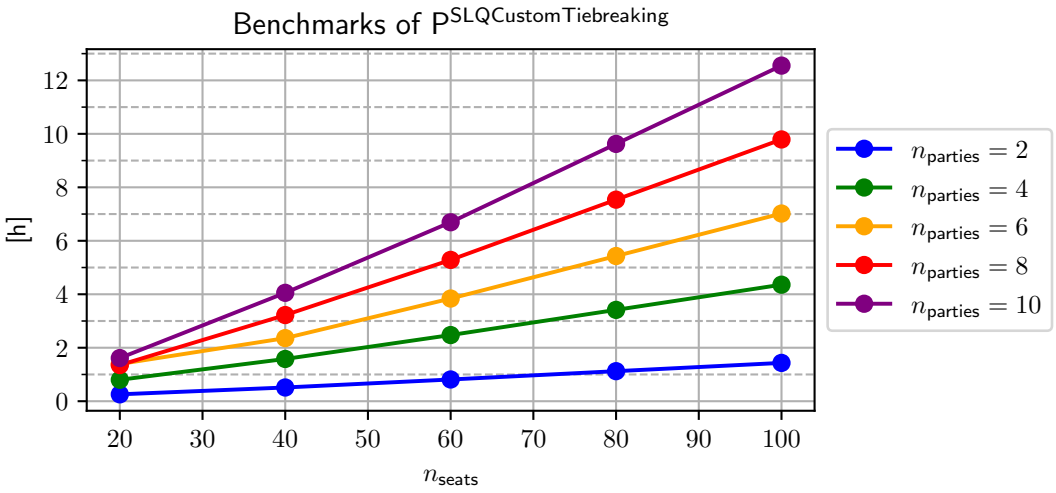


Figure 3.23.: Benchmarks of Ordinos evaluating $\mathsf{P}^{\mathsf{SLQBasic}}$ with $n_{\mathsf{trustees}} = 12$.

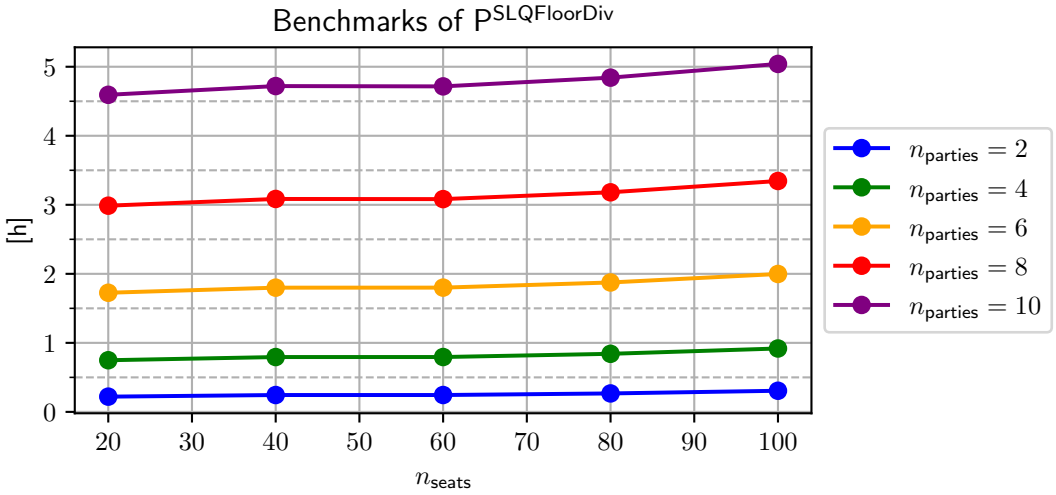


Figure 3.24.: Benchmarks of Ordinos evaluating $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ with $n_{\mathsf{trustees}} = 12$.

used an upper bound of n_{parties} iterations in our MPC protocol, ensuring that the number of iterations required remains hidden.

We present benchmarks in Figures 3.23 and 3.24. These benchmarks show that this variant of the Sainte-Laguë method is indeed faster than $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ for larger numbers of seats and smaller numbers of parties. For example, we have the following runtime for ten parties: To distribute 60 or 100 seats using $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$, the runtime is $6.7h$ or $12.6h$ respectively, while $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ only needs $4.7h$ or $5h$ respectively. However, for smaller numbers of, say, 20 seats, $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ is faster with $1.6h$ instead of $\mathsf{P}^{\mathsf{SLQFloorDiv}}$, which needs $4.6h$. $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ is linear in the number of seats while $\mathsf{P}^{\mathsf{SLQFloorDiv}}$ has a larger initial overhead time but is nearly constant in the number of seats.

3.2.3.21. Security

In this section, we prove the security of our instantiations of the Ordinos framework.

Theorem 3.6 (Security of the Ordinos Instantiations)**.** *Let $\mathcal{E}$ be an additively homomorphic IND-CPA-secure t-out-of-n_{trustees} threshold public-key encryption scheme and $\pi^{\mathsf{KeyShareGen}}$ be a secure NIZKP for $\mathcal{E}$ such as, e.g., the primitives presented above. Let $\pi^{\mathfrak{C}}$ be the choice space NIZKP, and let $\mathsf{P}^{f^{\text{res}}}$ be an MPC component as presented above. Then, the* Ordinos *instance using these primitives is an accountable and private (and hence tally-hiding) voting system for the election result function f^{res}.*

Proof. This theorem is a direct corollary of Theorems 3.2 to 3.4 which were proven in [Mül19]. Observe that the primitives $\mathcal{E}$, $\pi^{\mathsf{KeyShareGen}}$, and $\pi^{\mathfrak{C}}$ already fulfill the requirements of Theorems 3.2 to 3.4. The only thing left to show for Theorems 3.2 to 3.4 is that our new tallying protocol $\mathsf{P}^{f^{\text{res}}}$ is secure. That is, we have to show that $\mathsf{P}^{f^{\text{res}}}$ is a private and publicly accountable implementation of f^{res}.

Both properties follow because we constructed our MPC protocols from combinations of the basic components presented above. As mentioned in that section, these basic components guarantee privacy and public accountability. As for the connections of these components, the respective inputs and outputs are all encrypted (except for the final decryption of the election result) and published on the bulletin board. Due to the encryption, these intermediate results do not leak any additional information to internal parties or external observers. Also, since the intermediate results are published, external observers can check that the output of one step is used correctly as the input to the next step. Thus, if some trustee tries to use a different input, this trustee can be held accountable. □

3.2.3.22. Real-World Elections with Ordinos

This section presents benchmarks for the Ordinos instantiations of real-world elections. These benchmarks only evaluate the election result function, not including the ballots and ZKPs demonstrating their validity. In some cases, the ZKPs influence the overall system's runtime, such as the ZKPs for $\mathfrak{C}_{\mathsf{RankingMatrix}}$ presented in [HPT19], which requires trustees to compute, and is cubic in the number of candidates.

House of Commons with Ordinos*.* The elections for the House of Commons use $f^{\text{res}}_{\mathsf{Plurality}}$ in each of the 650 constituencies. Even for 200 candidates in a single constituency, the trustees can evaluate $f^{\text{res}}_{\mathsf{Plurality}}$ efficiently in under two hours for an arbitrary number of voters.

Elections for the Fachkollegien in the Deutsche Forschungsgesellschaft with Ordinos*.* These elections use $f^{\text{res}}_{\mathsf{RankingVotesBest},n}$ with up to 32 candidates. Ordinos handles 32 candidates for this election result function in under two hours for arbitrary numbers of voters.

Parliamentary Elections in the Republic of Nauru with Ordinos*.* To support the Dowdall system used by the parliamentary elections in the Republic of Nauru, we scale the points such that they are natural numbers. Since there are no more than 18 candidates, we scale the points

by $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 = 510,510$, which means that a voter can give at most $510,510$ points to a candidate. Knowing that there are no more than $1,630$ voters in a constituency, a candidate can receive at most $832,131,300$ points, which we can represent with 32 bits. Therefore, we can efficiently evaluate the parliamentary elections using $f^{\text{res}}_{\mathsf{Plurality}}$ in the Republic of Nauru with Ordinos since there are 18 candidates at most.

Debian Project with Ordinos. The Debian Project conducts elections using $f^{\text{res}}_{\mathsf{CondorcetSchulze}}$. The elections for the DPL typically have up to four candidates, which Ordinos evaluates in less than half an hour for up to 2^{64} voters. Other Debian Project elections have more choices than four, for example, seven, which Ordinos evaluates in under three and a half hours, and even faster for lower thresholds and bit sizes. Thus, Ordinos can efficiently evaluate $f^{\text{res}}_{\mathsf{CondorcetSchulze}}$ for elections of the Debian Project.

The New South Wales Legislative Assembly with Ordinos. Figure 3.21 shows that Ordinos evaluates $f^{\text{res}}_{\mathsf{IRVNSWPartial}}$ for up to five candidates in about two hours and six candidates in about twelve hours. Therefore, Ordinos can evaluate some of the districts of the election. However, as many (choice) components exist, the ZKPs become inefficient for more than five candidates and are the limiting factor for this election using the Ordinos system. Realistically, using specialized ZKPs from [CDS94, SV15], we expect that Ordinos supports up to five candidates for $f^{\text{res}}_{\mathsf{IRVNSWPartial}}$.

The Maine House of Representatives with Ordinos. As described above, Ordinos supports up to five candidates for $f^{\text{res}}_{\mathsf{IRVNSWPartial}}$ and therefore also for $f^{\text{res}}_{\mathsf{IRVLotPartial}}$, making it suitable for the Maine House of Representatives, as well as the US presidential and senatorial elections in Maine.

Elections for the German Bundestag with Ordinos. The election for the German Bundestag uses a complex system. In particular, it uses multiple constituencies. Since the Ordinos framework, as defined in Section 3.2.1 does not cover constituencies, we have to adapt the framework. We do so in Section 3.2.4 and present our Ordinos instantiation of the election for the German Bundestag there.

3.2.4. Voting in Multiple Constituencies: Evaluating the Elections for the German Bundestag

We designed the Ordinos framework presented above for elections without electoral constituencies or with just a single constituency where we treat all votes equally. For multi-constituency elections like the German parliamentary election, we have to make minor modifications to account for constituency-specific votes. During the setup phase, the scheduler publishes the list of eligible voters and assigns each to a constituency. Ballots now include the constituency identifier, allowing observers to check if voters voted for the correct constituency. Encrypted ballots are then aggregated per constituency and evaluated using the MPC component for f^{res}.

Employing multiple constituencies also slightly changes the meaning of full tally-hiding: For elections without electoral constituencies, we only reveal the number of submitted votes (since this is public on BB) and the final result. In the setting with electoral constituencies, we reveal

the number of submitted votes *per constituency* and the final result. In the following, we construct an MPC protocol $\mathsf{P}^{\mathrm{GermanBT}}$ that computes the election result function of the German Bundestag (see Section 2.8). Since this protocol uses multiple constituencies, as part of the security proof of $\mathsf{P}^{\mathrm{GermanBT}}$ (see Theorem 3.7), we define full tally-hiding for this setting and verify that the original proofs of Theorems 3.2 to 3.5 naturally carry over to this setting using the same preconditions.

We instantiate the (modified) Ordinos approach for the German election by using the threshold Paillier encryption scheme $\mathcal{E}$, choice space $\mathfrak{C}_{\mathsf{Single}} \times \mathfrak{C}_{\mathsf{Single}}$, standard NIZKPs $\pi^{\mathsf{KeyShareGen}}$ and $\pi^{\mathfrak{C}_{\mathsf{Single}}}$ [DJN10] and any standard EUF-CMA-secure signature scheme from the literature, result function $f^{\mathrm{res}}_{\mathsf{GermanBT}}$ for the German parliamentary election as described above, and notably our new MPC protocol $\mathsf{P}^{\mathrm{GermanBT}}$ for $f^{\mathrm{res}}_{\mathsf{GermanBT}}$ described next.

3.2.4.1. Constructing $\mathsf{P}^{\mathsf{GermanBT}}$

We construct $\mathsf{P}^{\mathrm{GermanBT}}$ using the components from Section 3.2.3 to compute the complete evaluation procedure for the German parliament presented in Section 2.8. The evaluation includes all diminutive details and exceptional cases, e.g., computing and changing the final parliament size, determining and distributing up to 3 overhang seats per party, and exempting parties from obtaining state seats if they did not win 5% of the total second votes, won less than 3 direct mandates, and are not representing a unique minority.

Of course, capturing the full complexity of the election evaluation of the German parliament in an MPC protocol $\mathsf{P}^{\mathrm{GermanBT}}$ comes at a massive cost in terms of performance and hence runs the risk of becoming impractical. Therefore, we have carefully optimized $\mathsf{P}^{\mathrm{GermanBT}}$ by, among others, the following:

- Computing the election result requires multiple iterations of the Sainte-Laguë method. We use $\mathsf{P}^{\mathsf{SLQCustomTiebreaking}}$ and $\mathsf{P}^{\mathsf{SLQFloorDiv}}$, depending on the seats and candidates in the current iteration.
- We have constructed $\mathsf{P}^{\mathrm{GermanBT}}$ so that we can compute substeps such as repeated state-wise operations in parallel. We have performed benchmarks for various threads, demonstrating that this is a significant factor in improving performance; see Table 3.1.
- We first compute and reveal the parties that will obtain at least one seat in the parliament. Disclosing this information allows us to tailor the following computations to this specific set of parties and thus save time by not having to perform the same operations for (dozens of) parties that will not obtain a seat. As part of Theorem 3.7, we argue that this construction is still a secure MPC protocol as, intuitively, the intermediate output is part of the final result.
- By proposing a different algorithm for computing the final number of seats for each party in the German parliament based on an encrypted divisor $d = \min(d_{\mathrm{no}}, d_{\mathrm{overh}})$, we can employ an efficient binary search on encrypted data to obtain this value.

Algorithm 3.24: $\mathsf{P}^{\mathsf{TestSeatsOfParty}}$

Input: n^j_{seats}, $E_{\mathsf{pk}}(n^j_{\mathsf{votes}})$, $E_{\mathsf{pk}}(d)$

1 $E_{\mathsf{pk}}(q) = \text{CreateEncFraction}(2 \odot E_{\mathsf{pk}}(n^j_{\mathsf{votes}}), 2 \cdot n^j_{\mathsf{seats}} - 1)$

2 $E_{\mathsf{pk}}(i) = f_{\mathsf{gt}}(E_{\mathsf{pk}}(q), E_{\mathsf{pk}}(d))$

3 **return** $1 - f_{\mathsf{gt}}(E_{\mathsf{pk}}(q), E_{\mathsf{pk}}(d))$

Regarding the last point described above, we can compute a ciphertext of the divisor d, which we use to compute the final number of seats in the German parliament and the number per party. To determine how many seats c_j^{party} receives, we would then – as in the suitable-denominator algorithm – compute $n^j_{\mathsf{seats}} = \lfloor \frac{n^j_{\mathsf{votes}}}{d} \rceil$. However, to compute this in a tally-hiding way, we would have to perform a floor division, followed by suitable rounding, which is non-trivial on encrypted values. Instead, we use that we can reveal the final number of seats per party since this is public information (we will discuss this in more detail in the security proof of Theorem 3.7). We construct an MPC protocol to check whether a party exceeds its given seats in a correct seat distribution based on d. Then, we can perform a public binary search and use this MPC component to find the correct number of seats. To realize such a checking functionality, we use the following: Consider an execution of the highest-quotients algorithm for distributing n_{seats} seats, where $q_{n_{\mathsf{seats}}}$ is the n_{seats}-th highest quotient and $q_{n_{\mathsf{seats}}+1}$ is the next highest quotient. Then it holds that a suitable denominator d of an execution of the equivalent suitable-denominator algorithm is in the range $(2 \cdot q_{n_{\mathsf{seats}}+1}, 2 \cdot q_{n_{\mathsf{seats}}}]$ [BTP06]. Therefore, if a quotient q causes a seat assignment in the highest-quotient algorithm, then the suitable-denominator d of the equivalent suitable-denominator algorithm must be of size at most $2 \cdot q$. We use this to construct the algorithm TestSeatsMatchDivisor presented in Algorithm 3.24 that for c_j^{party} takes as input a guess of seats n^j_{seats}, the number of votes $E_{\mathsf{pk}}(n^j_{\mathsf{votes}})$, and the divisor $E_{\mathsf{pk}}(d)$ and checks whether the divisor is less or equal to $\frac{2 \cdot n^j_{\mathsf{votes}}}{2 \cdot n^j_{\mathsf{seats}} - 1}$. If this is not the case, then this divisor d does not assign the seat of this quotient to this party. Then, we update the guess n^j_{seats} and try again. In Algorithm 3.25, we use TestSeatsMatchDivisor in a binary search.

Theorem 3.7 (Security of $\mathsf{P}^{\mathrm{GermanBT}}$)**.** *Let* $\mathsf{P}^{GermanBT}$ *be our MPC protocol from above. Then, the* Ordinos *instance using the abovementioned components is an accountable (and therefore also verifiable) and fully tally-hiding e-voting system for the election of the German parliament.*

Proof. We first observe that the design of the original Ordinos framework presented in Section 3.2.1 considers monolithic elections, which reveal nothing beyond the election result and the total number of votes. The election for the German Bundestag is not monolithic but instead based on multiple sub-elections in separate constituencies. We reflect this difference in the adapted Ordinos framework, which publishes the total number of submitted votes *for each constituency* (instead of just the total number of all votes) and the final election result. Hence, privacy/full tally-hiding has a slightly different meaning in our setting, and existing proofs for Ordinos do not directly apply.

Algorithm 3.25: $\mathsf{P}^{\mathsf{FindFinalSeatsPerParty}}$

Input: $(E_{\mathsf{pk}}(n^j_{\mathsf{votes}}))_{j=1}^{n_{\mathsf{parties}}}$, $E_{\mathsf{pk}}(d)$

for $j \in \{1, \ldots, n_{\mathsf{parties}}\}$ **do**
  $i = 1$
  $h = 0$
  while $i == 1$ **do** ▷ Find hundreds
    $i = \mathsf{P}^{\mathsf{TestSeatsOfParty}}(h, E_{\mathsf{pk}}(d), E_{\mathsf{pk}}(n^j_{\mathsf{votes}}))$
    if $i == 1$ **then**
      $h = h + 100$
  $l = h - 100$
  $u = h$
  while $l < u$ **do** ▷ Find remaining value via binary search
    $k = \lfloor l + \frac{1}{2}(u - l) \rfloor$
    $i = \mathsf{P}^{\mathsf{TestSeatsOfParty}}(k, E_{\mathsf{pk}}(d), E_{\mathsf{pk}}(n^j_{\mathsf{votes}}))$
    if $i == 1$ **then**
      $l = k$
    if $i == 0$ **then**
      $u = k$
    if $l + 1 = u$ **then**
      break
  $n^j_{\mathsf{seats}} = l$
return $(n^j_{\mathsf{seats}})_{j=1}^{n_{\mathsf{parties}}}$

For this reason, we modify the Ordinos framework in the following ways. During the setup phase, the scheduler publishes the list *id* of eligible voters and assigns each voter to a constituency. Ballots now include the constituency identifier, allowing observers to check if voters voted for the correct constituency. Encrypted ballots are then aggregated per constituency and evaluated using the MPC component for f^{res}. Furthermore, we modify the ideal protocol such that the voter under observation votes in a specific constituency.

These changes do not affect *accountability* (and thus *verifiability*), and therefore the original proofs from the original Ordinos framework for Theorems 3.2 to 3.5 still apply. However, it is public information to which constituency a ballot corresponds (the ballot contains the voter's id, which the scheduler published during the setup phase alongside the corresponding constituency). These changes naturally lead to a lower level of *privacy* than in the original Ordinos framework. While voting in constituencies might open new options for an adversary, as she can try to change the voting constituencies of specific ballots, she cannot do so without being caught. Thus, when considering risk-avoiding adversaries that aim not to be caught and only manipulate a fixed number of ballots, these adversaries would only manipulate up to a certain number of ballots overall (and not per constituency), since the global risk of being caught would be too high. Hence, to a certain extent, the privacy analysis from the original Ordinos framework carries over (with the natural restrictions of public constituency identifiers).

The above discussion allows us to prove security similar to the proofs of the original Ordinos framework. That is, if we can show that all of our cryptographic primitives, including P meet the requirements stated for Theorem 3.7 in [KLM+20a], then it follows analogous to the proof in [KLM+20a] that our system achieves

1. accountability and
2. privacy/full tally-hiding in the sense of our setting, i.e., only the final result and the total numbers of submitted votes for each constituency are revealed.

Next, observe that the primitives $\mathcal{E}$, $\pi^{\mathsf{KeyShareGen}}$, and $\pi^{\mathfrak{C}_{\mathsf{Single}}}$ already fulfill the requirements of Theorem 3.7. The only thing left to show for Theorem 3.7 is that our new tallying protocol P is secure. That is, we have to show that P is a private and publicly accountable implementation of the evaluation of the German Bundestag.

Both properties follow almost directly because we construct our MPC protocol using combinations of the components presented in Section 3.2.3. As mentioned in that section, these components already guarantee privacy and public accountability per component. Apart from the few exceptions discussed below, we have that the connections of these components, i.e., the respective inputs and outputs, are all encrypted and published on BB. Due to the encryption, these intermediate results do not leak any additional information to internal parties or external observers. Also, since the encrypted intermediate results are published, external observers can check that the output of one step is used correctly as the input to the next step. Thus, if some trustee tries to use a different input, she can be held accountable. Hence, such combinations of the individual components remain accountable and private.

The only exceptional cases and exceptions to the above are that we decrypt and reveal which parties enter the Bundestag and how many seats each party receives as an intermediate value to facilitate a more efficient evaluation of the following computations. However, this is still secure: observe that the election's final result maps seats to candidates. Since the mapping from candidates to parties is unique public knowledge, and candidates can only receive a seat if their party wins that seat, we can compute the seats assigned to each party from the final election result. In particular, given only the election result, a simulator for our MPC protocol can compute the corresponding intermediate values decrypted/revealed during a run of our MPC protocol. Given this information, the simulator can then use standard techniques as described in [DJN10, SV15] for simulating the decryption of ciphertexts to the correct intermediate values. □

3.2.4.2. Benchmarks of $\mathsf{P}^{\mathsf{GermanBT}}$

In this section, we present the benchmarks of our evaluation of the election for the German Bundestag in 2021. This election had 61,181,072 eligible voters, 46,854,508 valid submitted ballots, and 47 parties with 6,211 candidates distributed over 299 constituencies. We note

# Threads per Trustee	1	2	4	8	16	32
Single-member constituency seats	40.0 h	20.0 h	10.1 h	5.0 h	2.6 h	1.3 h
Determine which parties enter the Bundestag	71 min	36 min	18 min	9 min	5 min	3 min
First low distribution	23.4 h	11.8 h	6.1 h	3.3 h	1.8 h	1.1 h
Minimal number of seats per party	11.7 h	5.8 h	2.9 h	1.5 h	0.7 h	0.4 h
Second top distribution	2.8 h	2.0 h	1.4 h	1.2 h	1.2 h	1.2 h
Second low distribution	77.1 h	38.5 h	19.4 h	12.3 h	6.3 h	5.9 h
Assigning overhang seats	6.7 h	3.3 h	2.2 h	1.1 h	1.1 h	1.1 h
Computing the final result	4 min	2 min	1 min	1 min	0 min	0 min
Total Runtime	163 h	82 h	42 h	24.3 h	13.8 h	11.1 h

Table 3.1.: Benchmarks of the election for the German Bundestag in 2021 using real-world data available at [Der21]. Total runtime of the evaluation for the German parliament with real-world data from 2021 and different numbers of available parallel threads for each trustee.

that, for this, we have used the real-world data available at [Der21]. In Table 3.1, we show the benchmarks of our evaluation, including the individual benchmarks of all steps described in Section 2.8. We separate the benchmarks by the number of threads that each trustee uses. Since many intermediate results can be computed in parallel (e.g., the intermediate results of the states do not affect each other), allowing the trustees to employ multiple threads dramatically affects the overall runtime of the election evaluation. For example, as shown in Table 3.1, the runtime of the second low distribution can already be cut in half by using two threads instead of one. Our benchmarks indicate that the tally-hiding evaluation of the German parliamentary elections is practical for real-world election data.

3.2.5. Full-Fledged Tally-Hiding E-Voting with Ordinos

We have built a full-fledged, readily deployable web-based Ordinos system. Our system integrates all voting methods for which the Ordinos framework has been instantiated and benchmarked so far, including single vote, multiple vote, Borda, several Condorcet methods (plain Condorcet, weak Condorcet, Copeland, Smith set, several Minimax methods, and the Schulze method), and instant-runoff voting (IRV).

Our system uses Python3 for the backend, including the bulletin board and trustees, and the Quasar framework for the front end, which includes the voter programs, election authority, and authentication server.

Our implementation provides a graphical user interface in the web browser. The election authority can set up and manage several elections using different sets of trustees. The election authority can choose various options during the setup, like possible ballot formats and election result functions. The webpage checks that only possible combinations can be selected. When voters open the web page of an election, they can view a tutorial on how to vote and documentation on what security properties the system achieves. Voters cast their ballots using a web page,

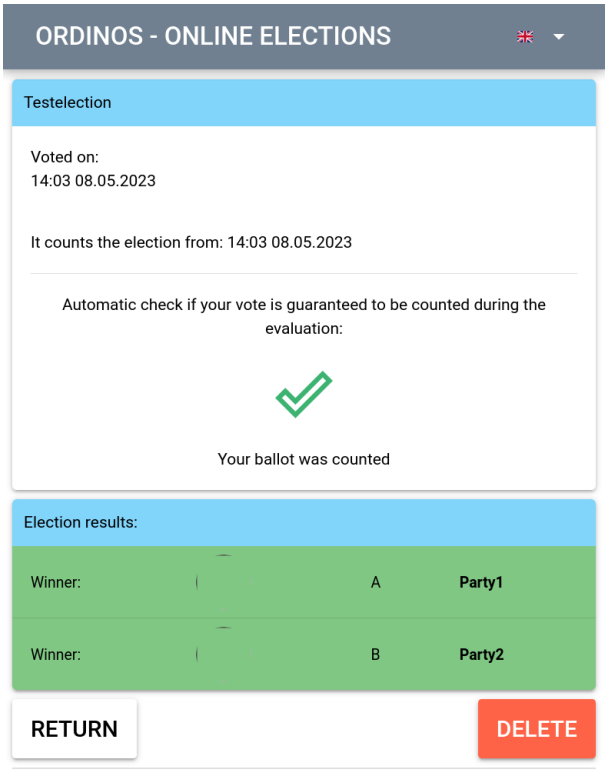


Figure 3.25.: The voter verification device of our web-based Ordinos system shows the election result.

which supports and encourages them to perform Benaloh challenges. We set up an Android application that can be used as the voter-verification device to check the correctness of the ballot creation process. The voter connects the Android application to the web page in the browser by scanning a QR code displayed in the web browser.

Furthermore, a web page shows the voter an overview of all participated elections, allowing the voter to see the results of evaluated elections. When the voter opens the results of an election, the Android application automatically starts the verification process of the election data, and it displays whether the election is verified or if any errors have occurred. Figure 3.25 shows a screenshot of an evaluated election. This automated verifiability process is similar to the one employed in [KMST16].

Our servers provide robustness: If a server crashes during the election, we can restart it and resume the election.

The implementation and documentation are available at [LAA+23a].

3.3. Related Work

Ordinos is the first provable secure, verifiable, and fully tally-hiding e-voting framework that can be instantiated for any election.

Previous proposals for fully tally-hiding e-voting systems, such as those by Benaloh [Ben86], Hevia and Kiwi [HK04], and Wen and Buckland [WB09], are limited in their applicability to a single specific voting method and lacked formal proofs of security, making them impractical solutions.

Szepieniec and Preneel proposed an e-voting protocol [SP15] that aims to compute the ranking of the candidates ($f^{\text{res}}_{\mathsf{Ranking}}$) or the candidate with the most votes ($f^{\text{res}}_{\mathsf{Plurality}}$) in a fully-tally hiding manner. However, the MPC protocol to compute the election result employs the Paillier encryption scheme and applies calculations that omit the modular reduction step. With that, the protocol leaks the most significant bits of plaintexts. One can prevent the leakage of sensitive data by ensuring that these bits do not contain any information. However, the MPC protocol adds random values to these plaintexts; there is always a chance that overflows occur and these leaked bits contain sensitive information about the tally. The authors acknowledge that it is impossible to detect all cases where such overflows occur in advance, e.g., before decrypting the ciphertexts. To tackle this issue, the authors introduce a *safety parameter* determining the probability of overflow cases occurring and how many of the most significant bits leak. We can ensure that such cases occur with very low probability by selecting appropriate values for the safety parameter. However, reducing the probability of overflows leads to more bits being leaked, drastically reducing the tally's confidentiality. Overall, the protocol proposed in [SP15] is not tally-hiding due to the leakage of sensitive information about the tally.

Canard et al. propose a fully tally-hiding e-voting system [CPST18] specifically for the full majority judgment evaluation ($f^{\text{res}}_{\mathsf{MJFull}}$, see Section 2.7). However, there is a non-negligible chance that the system does not output a result of an election's tally because the underlying evaluation algorithm does not provide a result for every possible tally. Canard et al. implemented their system and provide benchmarks demonstrating that it can handle large numbers of voters: Their system needs almost 20 minutes to tally 5 candidates with 5 possible grades and up to $2^{20} - 1$ voters using a single trustee. We did not instantiate Ordinos for the same election result function and thus can not compare the runtimes.

Cortier et al. [CGY22] propose universally verifiable MPC components for various election result functions. They design their MPC components for the exponential ElGamal cryptosystem instead of Paillier encryption, leading to differences in the runtime of basic operations compared to Ordinos, and, therefore, using different comparison protocols. Cortier et al. provide benchmarks for $f^{\text{res}}_{\mathsf{CondorcetSchulze}}$ using a single server with two 16-core AMD EPYC 7282 processors and 128 GB RAM. Unlike our Ordinos benchmarking setup, they do not use a threshold for the encryption scheme (that is, they set $t = 1$), use multi-threaded trustees, and their runtime increases with the number of voters. In comparison to Ordinos, their benchmarks indicate that their protocol can handle more significant numbers of candidates – however, under the restriction of lower numbers of voters: They present benchmarks for up to $1,024$ voters, and these benchmarks indicate that the runtime approximately doubles when doubling the number of voters. We anticipate that the Ordinos system will yield faster benchmarks for more significant numbers

of voters. To summarize, the work of Cortier et al. [CGY22] provides efficient benchmarks of $f^{\text{res}}_{\mathsf{CondorcetSchulze}}$ for small-scale numbers of voters, but the benchmarks indicate that the Ordinos system outperforms their system for more significant numbers of voters.

4. Partial Tally-Hiding: Ordinos

Fully tally-hiding e-voting systems use advanced cryptographic techniques to calculate the election results without revealing intermediate values. These systems rely on homomorphic cryptosystems to perform operations on the tally using only ciphertexts. However, these operations are usually expensive and time-consuming, and evaluating an election requires running them multiple times. Therefore, fully tally-hiding e-voting systems may take longer to compute the election results than participants are willing to wait.

Complex election result functions like instant-runoff voting (IRV) and single transferrable voting, an extension of IRV, pose a significant challenge for fully tally-hiding systems. Instant-runoff elections provide a poor level of privacy if they reveal the aggregated tally due to their large choice spaces. To improve privacy without employing full tally-hiding, research aimed to reveal intermediate values that allow for efficiency shortcuts in fully tally-hiding algorithms. These intermediate values reveal less information than the aggregated tally, which improves privacy. For instance, Ramchen et al. [RCPT19] proposed a system that computes IRV by revealing the first preference votes in each round. We refer to the behavior to reveal some parts of the tally as partial tally-hiding.

The Ordinos framework naturally supports partial tally-hiding by modifying the MPC protocol to output intermediate values. Based on these values, we can speed up the evaluation by breaking loops early and simplifying the control flow.

We organize this chapter as follows. We present the partial tally-hiding framework in Section 4.1. Section 4.2 presents partially tally-hiding instantiations of Ordinos. We discuss related work in Section 4.3.

4.1. Partially Tally-Hiding Framework

In this section, we introduce the notion of partially tally-hiding e-voting. Intuitively, an e-voting protocol P is partially tally-hiding for some voting method $(\mathfrak{C}, f^{\text{res}})$ if it computes $f_{\mathsf{intermediate}}$, which outputs intermediate values that we can use to compute f^{res}. Therefore, we require that P achieves the same level of privacy as the ideal voting protocol for evaluating $f_{\mathsf{intermediate}}$.

For $f_{\mathsf{intermediate}}$ given as above for a result function f^{res}, we define P to be $(f_{\mathsf{intermediate}}, f^{\text{res}})$-*partially tally-hiding* if it is fully tally-hiding w.r.t. $f_{\mathsf{intermediate}}$.

In the following, we analyze the impact of hiding the tally in a partial tally-hiding way.

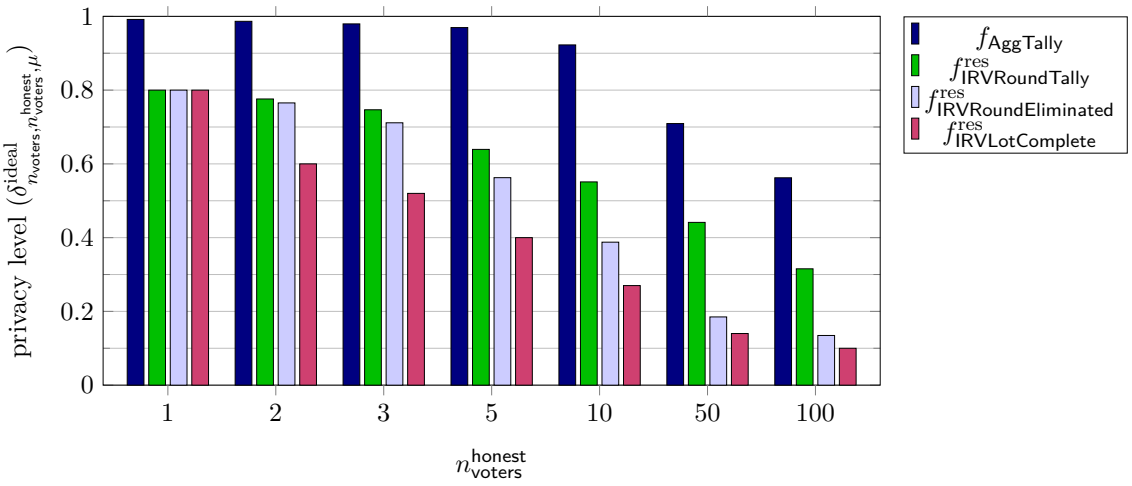


Figure 4.1.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for the ideal protocol with five candidates, IRV, uniform vote distribution, and no dishonest voters.

4.1.1. Impact of Hiding the Tally

In this section, we elaborate on the impact of hiding the tally for complex result functions using partially and fully tally-hiding approaches. We measure the privacy level for IRV using various (election result) functions. We present the privacy levels for IRV with five candidates and a uniform vote distribution in Figure 4.1. These values show that revealing the aggregated tally leads to almost no privacy. Publishing only the aggregated first preferences each round noticeably improves privacy because it hides the rankings and only reveals the currently first-ranked positions. As the figure shows, only revealing the eliminated candidate each round further improves the privacy level, which gets close to the ideal privacy level of $f^{\text{res}}_{\text{IRVLotComplete}}$ for more significant numbers of candidates.

In summary, evaluating $f^{\text{res}}_{\text{IRV}}$ in a partial tally-hiding way by computing $f^{\text{res}}_{\text{IRVRoundEliminated}}$ and thus achieving $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}(\mathfrak{C}_{\text{RankingPermutation}}, f^{\text{res}}_{\text{IRVRoundEliminated}})$-privacy slightly lowers the level of privacy, while allowing for a much more efficient evaluation, as we will see in Section 4.2.

4.2. Partial Tally-Hiding with Ordinos

As discussed and proven secure in Section 3.2, we can instantiate the Ordinos framework for multi-party computation protocols computing arbitrary functions. In particular, we can employ functions that compute the election result in a partially tally-hiding way. These partially tally-hiding MPC protocols aim to provide more efficient evaluations than their fully tally-hiding counterparts.

Loops are a significant factor for the runtimes of many of the fully tally-hiding protocols from Section 3.2.3. In particular, optimizing the number of overall iterations can drastically

optimize the overall runtime for complex election result functions with expensive runtimes for each iteration. We will investigate such optimizations for $f^{\text{res}}_{\mathsf{CondorcetSmithSet}}$.

As explained in Section 3.2.3, evaluating instant-runoff voting (IRV) using $\mathsf{P}^{f^{\text{res}}_{\mathsf{IRV}}}$ can be expensive due to the need to calculate the first preference of the ballots, which requires a large number of multiplications. However, we can improve this by modifying the computed function to reveal the eliminated candidate in each round. This way, we no longer need to perform these multiplications since the first preference of the rankings is already public knowledge. The following section describes how this modified function can significantly speed up the evaluation of IRV.

The following section discusses modifications to the Ordinos instantiations from Section 3.2.3 regarding $f^{\text{res}}_{\mathsf{CondorcetSmithSet}}$ and $f^{\text{res}}_{\mathsf{IRV}}$.

4.2.1. Condorcet

As discussed above, loops are a significant factor for the runtimes of many of the fully tally-hiding protocols from Section 3.2.3. An example of such a loop is in the MPC protocol for computing $f^{\text{res}}_{\mathsf{CondorcetSmithSet}}$: The protocol $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetSmithSet}}}$ computes the Copeland points of each candidate and then iterates through all possible Copeland points and iteratively adds the candidates having the current points to the resulting set until it satisfies a specific condition. The protocol must iterate through all possible Copeland points to ensure complete tally-hiding, regardless of whether it has already found the Smith set.

Applying partial tally-hiding, we propose the function $f^{\text{res}}_{\mathsf{CondorcetSmithSetPartial}}$ that computes the Smith set and the minimum Copeland points of the candidates in the Smith set. Computing this function with an MPC protocol allows us to compute $f^{\text{res}}_{\mathsf{CondorcetSmithSetPartial}}$-partially tally-hiding $f^{\text{res}}_{\mathsf{CondorcetSmithSet}}$. Computing the function $f^{\text{res}}_{\mathsf{CondorcetSmithSetPartial}}$ allows us to reveal the minimal Copeland points in the Smith set, and therefore, we can terminate the loop if the protocol finds the Smith set.

We present $\mathsf{P}^{f^{\text{res}}_{\mathsf{CondorcetSmithSetPartial}}}$ in Algorithm 4.1. This MPC protocol decrypts the variable f in each iteration. This variable indicates whether the Smith set is complete, and thus, if $f = 1$, we can terminate the loop over the Copeland points and decrypt the current set of candidates.

An interesting question is what the minimum number of Copeland points of the candidates in the Smith set is since this number directly determines the number of iterations the algorithm has to evaluate. We empirically measured this number for five candidates and two different voting distributions. We sampled $1,000,000$ tallies using a uniform vote distribution for various numbers of candidates and present the results in Figure 4.2. In comparison, we did the same for an extreme voting distribution. This extreme voting distribution assumes that the voters will likely prefer candidates representing related opinions. We model this by assigning the numbers $1, \ldots, n_{\mathsf{cand}}$ to the candidates and using a voting distribution that uses increased probabilities for rankings where candidates ranked next to each other have minimal distance regarding their numbers. For example, if a voter ranks the candidate with number three first, she is likelier to

Algorithm 4.1: $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSetPartial}}}$

Input: $E^{\text{Matrix}}_{\text{pk}}(\text{DM}^{\text{agg}})$

$E^{\text{Matrix}}_{\text{pk}}(D), E^{\text{Vector}}_{\text{pk}}(\text{d}^{\text{gteq}}), E^{\text{Vector}}_{\text{pk}}(\text{d}^{\text{gt}}) = \mathsf{P}^{\text{CondorcetHelper}}(E^{\text{Matrix}}_{\text{pk}}(\text{DM}^{\text{agg}}))$

$E^{\text{Vector}}_{\text{pk}}(b) = \text{CreateEncVector}(0, n_{\text{cand}})$ ▷ Initialize result vector

$E_{\text{pk}}(n) = E_{\text{pk}}(0)$

$E^{\text{Vector}}_{\text{pk}}(p^{\text{Copeland}}) = (E^{\text{Vector}}_{\text{pk}}(\text{d}^{\text{gt}})_i + E^{\text{Vector}}_{\text{pk}}(\text{d}^{\text{gteq}})_i)_{i=1}^{n_{\text{cand}}}$ ▷ Compute Copeland points

$E_{\text{pk}}(p^{\Phi}) = E_{\text{pk}}(0)$

for $t = 2 \cdot (n_{\text{cand}} - 1)$ ***downto*** 1 **do** ▷ Iterate over possible Copeland points

  for $i \in \{1, \ldots, n_{\text{cand}}\}$ **do**

    $E_{\text{pk}}(e) = f_{\text{eq}}(E^{\text{Vector}}_{\text{pk}}(p^{\text{Copeland}})_i, E_{\text{pk}}(t))$ ▷ Check Copeland points of candidate

    $E^{\text{Vector}}_{\text{pk}}(b)_i = E^{\text{Vector}}_{\text{pk}}(b)_i$

    $E_{\text{pk}}(n) = E_{\text{pk}}(n) \oplus E^{\text{Vector}}_{\text{pk}}(b)_i$ ▷ Update size of dominating set

    $E_{\text{pk}}(p^{\Phi}) = E_{\text{pk}}(p^{\Phi}) \oplus E^{\text{Vector}}_{\text{pk}}(b)_i$ ▷ Update Copeland points

  $E_{\text{pk}}(t_{\text{tmp}}) = E_{\text{pk}}(n) \odot (E_{\text{pk}}(2) \odot E_{\text{pk}}(n_{\text{cand}}) \ominus E_{\text{pk}}(n) \ominus E_{\text{pk}}(1))$

  $f = f_{\text{dec}}(f_{\text{eq}}(E_{\text{pk}}(p^{\Phi}), E_{\text{pk}}(t_{\text{tmp}})))$

  if $f == 1$ **then**

    return $(f_{\text{dec}}(E^{\text{Vector}}_{\text{pk}}(b)_i))_{i=1}^{n_{\text{cand}}}$

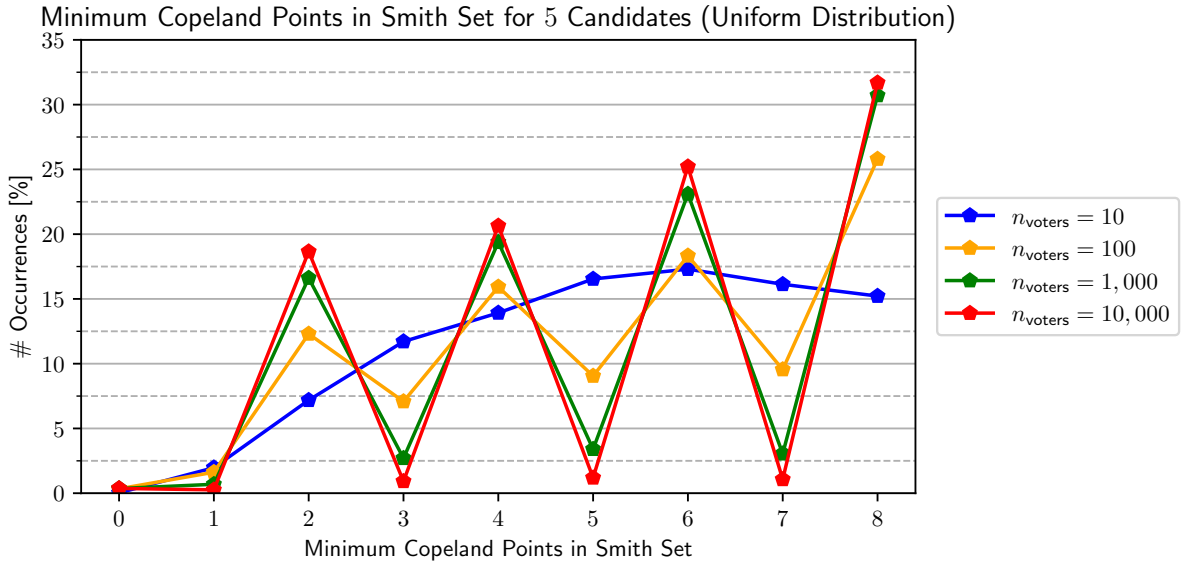


Figure 4.2.: Minimum Copeland points in Smith set for 5 candidates, uniform vote distribution, and $1,000,000$ samples.

assign candidate two or four the second rank. We present the results of this voting distribution in Figure 4.3.

While the minimum Copeland points in the Smith set vary using a uniform distribution, it is more likely for higher minimal Copeland points to occur than for lower ones. Switching to the extreme distribution, the average minimal number of Copeland points in the Smith set drastically increases. We present the expected values and the standard deviation in Table 4.1.

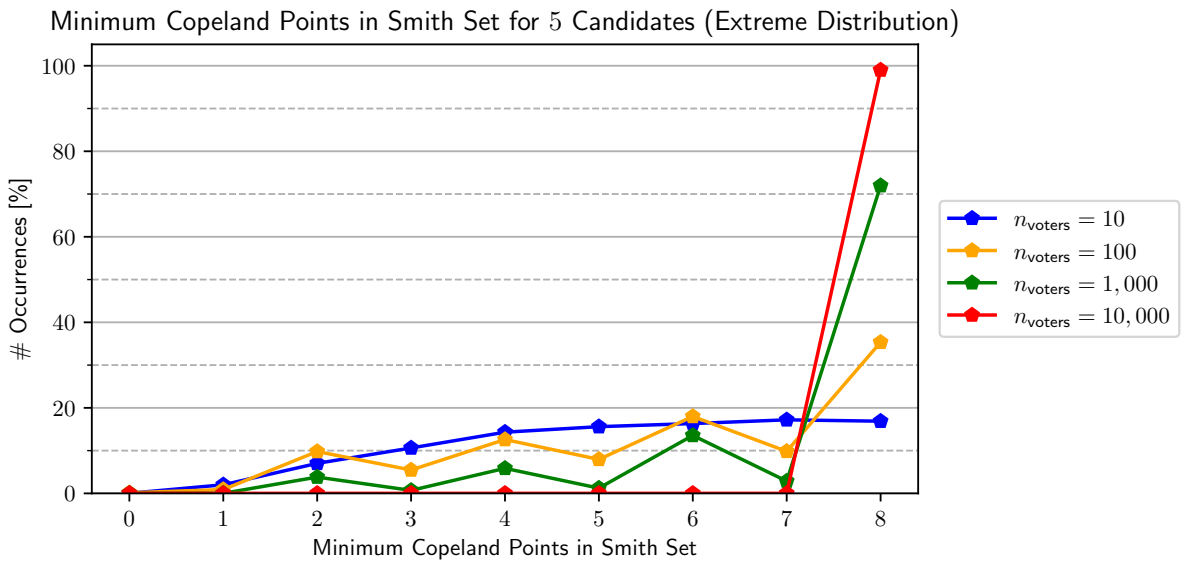


Figure 4.3.: Minimum Copeland points in Smith set for 5 candidates, extreme vote distribution, and $1,000,000$ samples.

n_{voters}	Uniform Distribution μ		Extreme Distribution μ	
	$E[X]$	σ	$E[X]$	σ
10	5.28	1.92	5.37	1.94
100	5.40	2.15	5.86	2.10
1,000	5.42	2.20	7.16	1.58
10,000	5.41	2.24	7.92	0.16

Table 4.1.: Expected minimum Copeland points in Smith set, $E[X]$ denotes the expected value and σ the standard deviation.

These values show that using the partially tally-hiding approach is more efficient than the fully tally-hiding approach.

We benchmarked the protocol $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSetPartial}}}$ in comparison with $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSet}}}$ for various numbers of candidates as follows. We evaluated three tallies using a uniform distribution, and Figure 4.4 presents the average runtime. This figure indicates that $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSetPartial}}}$ is more efficient than $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSet}}}$.

An interesting question is the impact regarding privacy: The partial tally-hiding MPC protocol $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSetPartial}}}$ reveals the minimum Copeland points in the Smith set. Therefore, we computed the privacy values of $f^{\text{res}}_{\text{CondorcetSmithSet}}$ and $f^{\text{res}}_{\text{CondorcetSmithSetPartial}}$ compared to f^{agg}. We present these values in Figure 4.5. The values reveal that $f^{\text{res}}_{\text{CondorcetSmithSetPartial}}$ has a privacy level equal to $f^{\text{res}}_{\text{CondorcetSmithSet}}$.

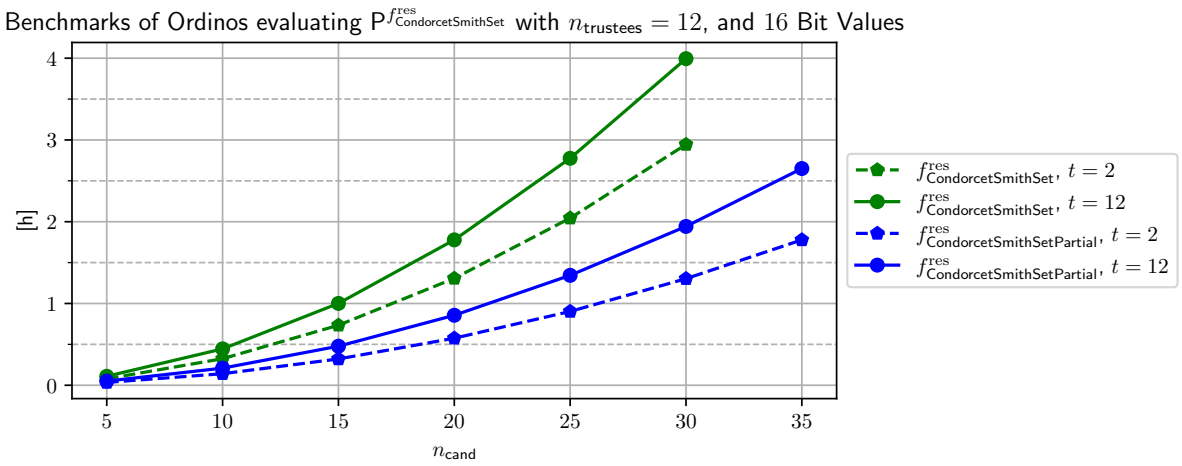


Figure 4.4.: Average benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSetPartial}}}$ in comparison to $\mathsf{P}^{f^{\text{res}}_{\text{CondorcetSmithSet}}}$ with $n_{\text{trustees}} = 12$, values of 16 bit, and three samples.

Algorithm 4.2: $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$

Input: $E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})$, X

$E^{\text{Vector}}_{\text{pk}}(v^{\text{round},i}) = \text{CreateEncVector}(0, n_{\text{cand}})$ ▷ Votes received in this round.

for $c \in \{1, \ldots, n_{\text{cand}}\}$ **do**

  for $j \in \{1, \ldots, n_{\text{components}}\}$ *s.t.* j *represents a ranking where the first preference is* c^{cand}_c *with respect to* X **do**

    $E^{\text{Vector}}_{\text{pk}}(v^{\text{round},i})_c = E^{\text{Vector}}_{\text{pk}}(v^{\text{round},i})_c \oplus E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})_j$

return $f_{\text{dec}}((E^{\text{Vector}}_{\text{pk}}(v^{\text{round},i})_c)_{c=1}^{n_{\text{cand}}})$

4.2.2. Instant-Runoff Voting

As stated in Section 3.2.3, the primary bottleneck in computing instant-runoff voting (IRV) with $\mathsf{P}^{f^{\text{res}}_{\text{IRV}}}$ is the time-consuming process of evaluating the current first preference of the ballots, which involves a significant number of multiplications that scales factorially in the number of candidates. However, a solution to this issue involves modifying the computed function to reveal the eliminated candidate in each round. By doing so, there is no need to perform these multiplications since the first preference of the rankings is public knowledge at that point.

We can adapt $\mathsf{P}^{f^{\text{res}}_{\text{IRV}}}$ to output the first preferences each round, allowing us to compute which candidate to eliminate in plain, realizing $f^{\text{res}}_{\text{IRVRoundTally}}$. With the knowledge of eliminated candidates, we know the current first preference of each possible ranking. Thus, we do not need to compute first preferences based on the ciphertexts. Therefore, the MPC protocol $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ does not include a multiplication per ranking per candidate, as in the case of $\mathsf{P}^{f^{\text{res}}_{\text{IRV}}}$. We present $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ in Algorithm 4.2.

The MPC protocol $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ does not include tie-breaking: We compute the candidate to eliminate in each round over the plain tally. We can include tie-breaking in the MPC protocol

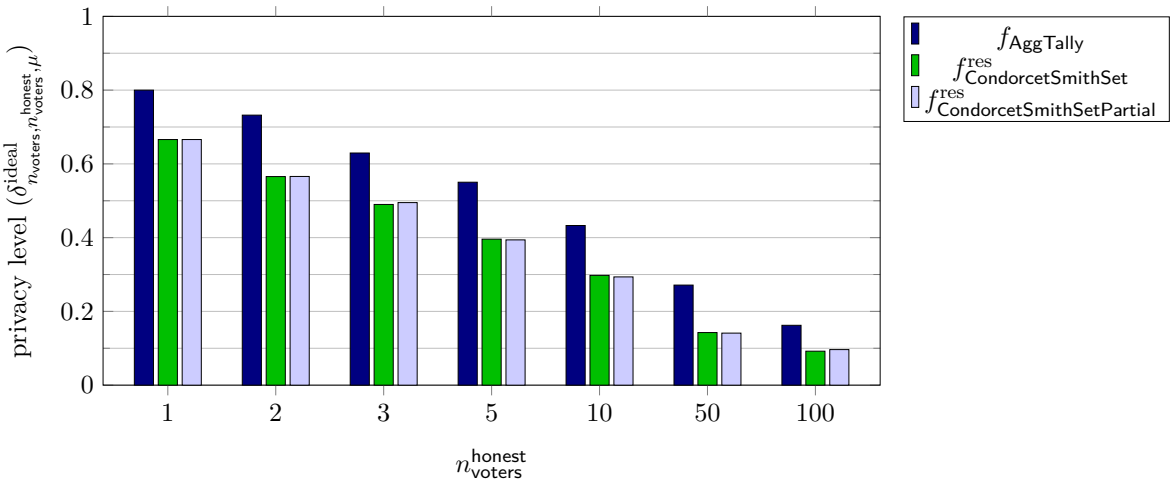


Figure 4.5.: Level of privacy $\delta^{\text{ideal}}_{n_{\text{voters}},n^{\text{honest}}_{\text{voters}},\mu}$ for the ideal protocol with three candidates, uniform vote distribution, and no dishonest voters.

Algorithm 4.3: $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundEliminated}}}$

Input: $E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})$, X

1 $E^{\text{Vector}}_{\text{pk}}(v^{\text{round},i}) = \mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}(E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}}), X)$

2 $E^{\text{Vector}}_{\text{pk}}(l) = \text{GetLastTieBreaking}(E^{\text{Vector}}_{\text{pk}}(X))$

3 **return** $f_{\text{dec}}((E^{\text{Vector}}_{\text{pk}}(l)_i)_{i=1}^{n_{\text{cand}}})$

and thus compute $f^{\text{res}}_{\text{IRVRoundEliminated}}$. We present $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundEliminated}}}$ in Algorithm 4.3. The runtime of $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundEliminated}}}$ depends on the specific tie-breaking mechanism, in contrast to $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$.

Figure 4.6 presents the benchmarks of $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ and of $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundEliminated}}}$ for various tie-breaking mechanism evaluating a single round and a complete evaluation of IRV. Since the number of cryptographic operations does not scale in the number of possible rankings, there is no noticeable difference between complete and partial rankings. Furthermore, $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ consists of a single decryption per candidate per round. Since decryptions are efficient operations, the difference between the runtime of a complete evaluation of IRV and a single round evaluation of $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ is not noticeable.

The benchmarks show that the runtimes of $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ and $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundEliminated}}}$ with tie-breaking by lot only differ up to a few minutes.

The figure also shows the complexity of the tie-breaking mechanism, which is not noticeable using full tally-hiding due to the factorial scaling in the number of candidates.

As discussed in Section 3.2.3, we expect that the ZKPs for $\mathfrak{C}_{\text{RankingMatrix}}$ supports five candidates for partial rankings. Thus, the ZKPs are the bottleneck for partial tally-hiding with Ordinos.

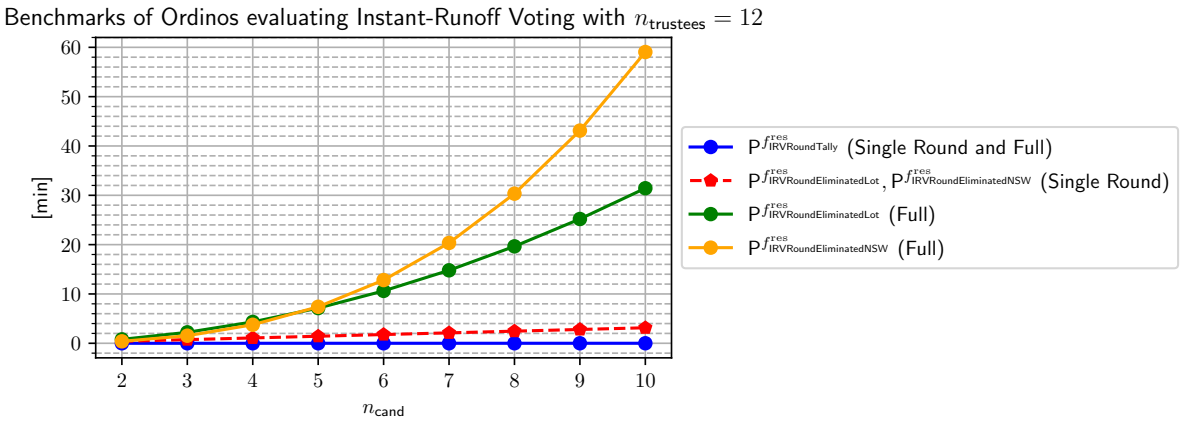


Figure 4.6.: Benchmarks of Ordinos evaluating $\mathsf{P}^{f^{\text{res}}_{\text{IRV}}}$ with $n_{\text{trustees}} = 12$ and $t = 12$. For comparison, see Figure 3.21 for benchmarks of the fully tally-hiding instantiations of Ordinos for the same result functions.

4.3. Related Work

Ramchen et al. proposed a partially tally-hiding system for $f^{\text{res}}_{\text{IRVNSWPartial}}$ in [RCPT19], which computes $f^{\text{res}}_{\text{IRVRoundTally}}$ and thus reveals the current first votes per candidate after each instant-runoff round. As we have shown in Figure 4.1, $f^{\text{res}}_{\text{IRVRoundTally}}$ improves the privacy level of f_{AggTally}, but is far away from $f^{\text{res}}_{\text{IRV}}$ and $f^{\text{res}}_{\text{IRVRoundEliminated}}$.

Ramchen et al. use a different choice space than $\mathfrak{C}_{\text{Single}}$ and do not specify zero-knowledge proofs for this choice space. It remains unclear how voters can prove the validity of their ballots. Additionally, the system does not aggregate the ballots but processes and updates them individually in each round. Thus, the evaluation of their system scales in the number of voters. In comparison, the Ordinos framework aggregates the ballots; therefore, the complexity does not scale in the number of voters.

They used an Intel i7-6770HQ processor with four cores (8 threads) and 32 GB of RAM. Their benchmarks did not include proof creation and validation, and revealing intermediate tallies allowed for early termination if the election winner was found. In comparison, we obtained our benchmarks using an ESPRIMO Q957 (64bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM) without parallelism and just a single core.

The benchmarks they evaluate are based on the election for the districts of Albury and Auburn in 2015. The election for the district of Albury had five candidates and required only a single round, taking them 116 minutes to evaluate. The election for the district of Auburn had five candidates, took four rounds, and was evaluated in approximately 15 hours. As shown in Figure 4.6, the fully tally-hiding instantiation of Ordinos evaluates Albury in under two hours and Auburn in about twelve hours for up to 2^{64} voters. The partial tally-hiding instantiation of Ordinos using $\mathsf{P}^{f^{\text{res}}_{\text{IRVRoundTally}}}$ evaluates both scenarios in under three minutes, and we can drastically

improve privacy by evaluating $f^{\text{res}}_{\mathsf{IRVRoundEliminated}}$, which Ordinos can handle in under 15 minutes for up six candidates.

Overall, our partially tally-hiding instantiations outperform Ramchen et al. [RCPT19] regarding privacy properties and performance.

5. Public Tally-Hiding: Kryvos

All current e-voting systems that hide the tally aim to keep it hidden from everyone, including voting authorities. In contrast, real-world elections often follow a different common practice for hiding the tally: In many cases, voting authorities choose to compute but not publish the aggregated tally. Instead, they publicize the final election result, e.g., the winner of the election or the n best candidates, as they wanted to mitigate privacy issues for voters to prevent, e.g., Italian attacks, for candidates to prevent, e.g., embarrassment and weak mandates, and manipulations, e.g., gerrymandering. Hence, while the trustees learn the aggregated tally, the public learns only the election result. We call this technique *public tally-hiding*. Such publicly tally-hiding elections are carried out, among others, by ACM Special Interest Groups (SIGs) [ACM20], the German Computer Science Association [Ges19], CrossRef [Cro19], the Society for Industrial and Applied Mathematics (SIAM) [Soc19], or the German Research Fund [Deu19]. Civica Election Services (CES) [Civ23], a large e-voting provider, conducts several dozen elections per year where customers demand to obtain only the actual election result from CES [Per20], according to CES, for example, to protect against weak mandates or gerrymandering issues. Although such elections are pretty standard, they do not offer verifiability, i.e., it is impossible for a voter or external observer to verify that the voting authority computed the election result correctly.

Public tally-hiding, motivated by and enhancing these existing practices, offers a trade-off between privacy and efficiency, which differs from all previous fully and partially tally-hiding solutions. Particularly, while publicly tally-hiding protocols hide the (aggregated) tally from the public, partially tally-hiding systems still reveal some intermediate information about the tally to the public.

We propose the first *verifiable* voting system that follows this common practice of publicly tally-hiding elections. More specifically, we present a publicly tally-hiding e-voting system that follows a radically different approach than all previous tally-hiding schemes. We allow each trustee to learn the aggregated tally while the public merely learns the final election result.

Applying public tally-hiding allows for a novel design of e-voting systems. We can use dissimilar cryptographic techniques compared to prior systems for tally-hiding elections. For example, we can employ lightweight ZKPs instead of more heavy-weight universally verifiable MPC. Our publicly tally-hiding e-voting system achieves practical efficiency for dramatically more significant numbers of candidates and more complex voting methods than all previous fully tally-hiding systems while still hiding the aggregated tally from the public, unlike partially tally-hiding systems. However, public tally-hiding comes with the trade-off that now the trustees learn the aggregated tally. Altogether, this work opens a new line of research enhancing already existing

practices, where voting authorities compute but do not publish the tally with the fundamental and crucial property of verifiability.

The design of Kryvos differs from previous (fully) tally-hiding e-voting systems. It builds on general-purpose ZKPs, specifically the highly efficient succinct non-interactive argument of knowledge (SNARK) by Groth16 [Gro16], which we call the Groth16 SNARK in this thesis. The core idea is that trustees prove the correctness of the election outcome according to the result function using this SNARK. While the concept itself is straightforward, putting this theory into practice requires solving several challenges:

- Kryvos follows the general design philosophy of Helios (see Section 2.1), one of the most prominent and practical verifiable e-voting systems that form the basis of many other systems. It is not possible to apply the above idea directly to Helios (see Section 5.2). Therefore, Kryvos is a fundamental redesign of Helios.

- To be of practical use, also with performance that enhances upon existing fully tally-hiding systems, the design of Kryvos requires cautiousness and dealing with several pitfalls (see Sections 5.2, 5.3.4, and 5.3.5). We evaluate various implementation and design possibilities to develop an appropriate implementation.

We provide instantiations of Kryvos for manifold voting methods, including plurality voting, various shapes of cumulative elections with multiple votes, and ranked elections, such as instant-runoff voting (IRV). We extensively evaluate these instantiations, verifying the practicality of the system. Among others, we test our system with real-world data of complex instant-runoff elections from Australia. Our benchmarks show that Kryvos is significantly more efficient than existing fully tally-hiding solutions: Kryvos takes under one minute to evaluate even complex elections.

Furthermore, utilizing general-purpose proof systems, Kryvos enables voters to cast ballots of complex choice paces. Even more, Kryvos efficiently handles choice spaces for which other ZKPs are inefficient. With that, we are the first to fully and efficiently support $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$ in an e-voting system, opening new possibilities for electronic voting. These ZKPs are not specific to public tally-hiding and can be employed in other e-voting systems.

As explained in the introduction, this chapter covers the content and is based on the publication [HKK+22a] and its technical report [HKK+22b], and some parts of this chapter are taken verbatim from them.

We organize this chapter as follows. In Section 5.1, we present the framework of public tally-hiding. Afterward, Section 5.2 presents the design rationale of Kryvos, the first provable secure verifiable and publicly tally-hiding e-voting system. Section 5.3 then presents the Kryvos system in detail. We discuss related publicly tally-hiding work in Section 5.4.

5.1. Publicly Tally-Hiding Framework

In this section, we introduce the novel notion of publicly tally-hiding e-voting. Intuitively, an e-voting protocol P is publicly tally-hiding for some voting method $(\mathfrak{C}, f^{\text{res}})$ if the following two conditions hold:

1. *Public privacy:* Under the assumption that all trustees are honest, the protocol P provides the same level of privacy as the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\text{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ for voting method $(\mathfrak{C}, f^{\text{res}})$, which reveals nothing but the actual election result by definition (see Section 3.1.1). Except for the trustees, no one learns anything beyond the published election result.

2. *Internal privacy:* Under the assumption that fewer trustees than a certain threshold t are dishonest, the protocol P provides the same level of privacy as the ideal voting protocol $\mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\text{res}}, n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu)$ for voting method $(\mathfrak{C}, f_{\mathsf{AggTally}})$, where f_{AggTally} is the function that returns the *aggregated tally* (e.g., the number of votes for all choices/candidates). In other words, the trustees do not learn more than they would for a non-tally-hiding secure e-voting protocol.

We now define this notion formally. We assume some set T of trustees and some threshold t.

Definition 5.1 (Publicly Tally-Hiding)**.** *Let* P *be a voting protocol with a set of trustees* T *and* $t \leq |\mathsf{T}|$*. We define that* P *is* $(\delta^{\text{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\text{res}}), \delta^{\text{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}}))$-publicly tally-hiding w.r.t. f^{res} and (T, t) *if (and only if) the following two conditions hold:*

Public Privacy: *If all parties of* T *are honest, then the protocol* P *achieves* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\text{res}})$*-privacy.*

Internal Privacy: *If at most* $t - 1$ *parties in* T *are dishonest, then the protocol* P *achieves* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$*-privacy.*

We call $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\text{res}})$ *the public and* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$ *the internal privacy level.*

As explained, a secure protocol's public and internal privacy levels should correspond to the privacy levels of the ideal protocols mentioned above.

5.1.1. Impact of Hiding the Tally

We measure the difference between internal and public privacy for $f^{\text{res}}_{\mathsf{Plurality}}$ and $f^{\text{res}}_{\mathsf{IRVLotComplete}}$ for various numbers of honest voters and five candidates. We present the results in Figure 5.1. These values show that the concept of public tally-hiding drastically improves privacy for the public for relatively simple and complex election result functions like $f^{\text{res}}_{\mathsf{IRVLotComplete}}$. The difference between the public and internal privacy levels further expands with more honest voters and non-uniform distributions, which is a typical setting of real-world elections. In summary, the

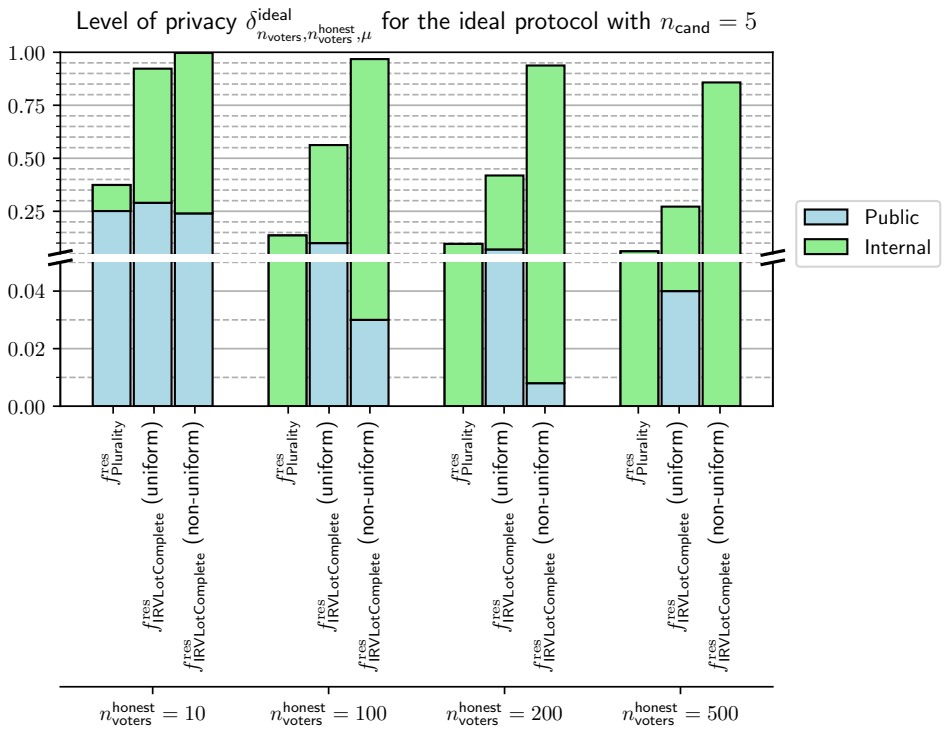


Figure 5.1.: Ideal internal and public levels of privacy for various election result functions.

concept of public tally-hiding drastically improves privacy for the public while still achieving the same privacy level as Helios for the trustees.

5.2. Design Rationale of Kryvos

This section describes the design choices that led to Kryvos, the first provable secure, verifiable, and publicly tally-hiding remote e-voting system.

Our concept for Kryvos is to construct a system similar to Helios where we publicly aggregate the encrypted votes, and the trustees compute $c^{\text{choice}}_{\text{agg}}$ from $E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})$. The aggregation guarantees trustees do not learn more about individual votes than trustees in a standard privacy-preserving (non-tally-hiding) system, such as Helios. However, rather than publicizing $c^{\text{choice}}_{\text{agg}}$, the trustees internally compute and publish solely the result $f^{\text{res}}(c^{\text{choice}}_{\text{agg}})$. To still obtain verifiability in this situation, we employ non-interactive zero-knowledge proofs (NIZKPs) to let (one of) the trustees prove that $f^{\text{res}}(c^{\text{choice}}_{\text{agg}})$ was computed correctly from $E^{\text{Vector}}_{\text{pk}}(c^{\text{choice}}_{\text{agg}})$. Since NIZKPs are relatively lightweight compared to multi-party computation, the rationale is that this should allow for assembling more efficient e-voting systems that support more types of elections in a (publicly) tally-hiding way than existing (fully) tally-hiding systems. However, traditional zero-knowledge proof systems (which we will call *specialized ZKPs* in the following) support one specific relation (e.g., the NIZKPs presented in Section 3.2.3 support exactly one choice space each). Consequently, applying such NIZKPs requires finding a secure and efficient NIZKP

for each election result function. We use another, more flexible class of ZKPs to support many existing election types with various result functions. We utilize recent advances in the domain of *general-purpose proof systems (GPPSs)*, which allow for proving statements for arbitrary functions.

A GPPS consists of the three algorithms ($\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}$):

1. The $\mathsf{Setup}(f_{\mathcal{R}})$ algorithm takes as input an arbitrary indicator function $f_{\mathcal{R}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ for some binary relation $\mathcal{R}$, such that $f_{\mathcal{R}}(x,w) = 1$ if (and only if) $(x,w) \in \mathcal{R}$ for public statement x and secret knowledge/*witness* w. This algorithm then outputs public parameters, typically in the form of two common reference strings, EKCRS (*evaluation key* CRS, needed to create proofs) and VKCRS (*verification key* CRS, needed to verify proofs) that depend on $f_{\mathcal{R}}$. The public parameters create an instance of the GPPS specific to the function $f_{\mathcal{R}}$.
2. Anyone can then use EKCRS to create a proof $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{EKCRS}, x, w)$.
3. To verify the proof, one can use $\mathsf{Verify}(\mathsf{VKCRS}, x, \pi)$.

Employing zero-knowledge GPPSs to prove that the declared election result corresponds to the tally's encryptions is not only advantageous to construct a publicly tally-hiding e-voting system. Moreover, the zero-knowledge GPPS is also efficient on devices with low computational power. Therefore, the voters can also utilize these to prove the validity of their ballots. Using GPPSs to verify ballot validity allows Kryvos to support any choice space, including complex ranked voting and point-based methods. This approach is practical, as we show in Section 5.3.5. This support for arbitrary ballot formats is also of interest beyond the area of (publicly) tally-hiding systems. For example, to our knowledge, Kryvos is the first e-voting system supporting ZKPs for Borda tournament-style ballots using $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$. For ranked ballots using $\mathfrak{C}_{\mathsf{RankingMatrix}}$, Kryvos improves over previous efficient ZKPs that even required trustees to perform part of the proof [HPT19].

While the general idea of making use of GPPSs might seem conceptually simple, it requires careful design and optimization to achieve not only a secure but also a practical solution as outlined next and in Sections 5.3.4 and 5.3.5.

5.2.1. Challenges

In order to construct an efficient publicly tally-hiding e-voting system based on the rationale described above, we have to overcome several challenges:

- The size of EKCRS: Regarding the proofs of showing the validity of the ballots, the size of the evaluation key common reference string (CRS) EKCRS must be as small as possible such that all voters, including users with limited resources, can create proofs and thus can participate in the election.

- The size of VKCRS: Overall, the size of the verification key common reference string (CRS) VKCRS must be as small as possible, such that everyone, including parties with limited resources, can store the VKCRS in order to verify proofs.
- The proofs should require less computational power and resources to verify. Large-scale elections with millions of votes require efficient verification of the ballots.
- The proofs should be as small as possible: Compact proofs require less storage space, making storing and managing voting data securely easier. Small proof size is critical when archiving election records for potential audits. Furthermore, it is essential to achieve small proof sizes while maintaining robust security and privacy protections. Smaller proofs benefit voters with limited resources, such as those using low-end devices or slow internet connections to cast their votes. Ensuring accessibility for all eligible voters is crucial to a democratic election.
- The proof generation algorithm should be as efficient as possible: Faster tallying times allow quicker and more efficient election results. Furthermore, voters must be able to efficiently create proofs showing the validity of their ballots, even when using devices with low computational power.
- Selecting a suitable cryptographic primitive: The GPPS must efficiently support the underlying cryptographic primitive used to hide the tally. As we will see later in this section, the cryptographic operations are the main complexity factor for the GPPS.

5.2.2. Selecting a Suitable Cryptographic Primitive

One major challenge of designing a verifiable publicly tally-hiding e-voting system by having trustees prove in zero-knowledge that $f^{\text{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ was computed correctly from $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ is that this approach requires combining the GPPS with cryptographic operations. As we will see later in this section, cryptographic operations are the main bottleneck of the efficiency of Kryvos. If we make use of a threshold encryption scheme, the GPPS would have to be computed locally by one of the trustees, who would require as witnesses all private key shares in order to prove knowledge of $c_{\mathsf{agg},w}^{\mathsf{choice}}$ such that $c_{\mathsf{agg},w}^{\mathsf{choice}} = (\mathsf{dec}(E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})_i))_{i=1}^{n_{\mathsf{components}}}$. Revealing those key shares would allow that trustee to decrypt individual votes, breaking ballot privacy entirely. Therefore, instead of decryption using the secret key, one would let $t-1$ trustees create decryption shares of every ciphertext of $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$, alongside proofs of correctness. Then, an additional trustee takes these $t-1$ decryption shares, alongside her secret key, as input for the GPPS, which now performs two steps to decrypt the tally:

1. The trustee, using her secret key, creates a decryption share of every ciphertext of $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$.
2. The trustee combines the t decryption shares to decrypt the ciphertext.

However, these steps require many cryptographic operations and are far too inefficient in an efficient publicly tally-hiding e-voting system. This work proposes a different approach:

constructing an e-voting system based on an additively homomorphic commitment scheme, following a line of commitment-based e-voting systems initiated by Cramer et al. [CFSY96]. The main difference to the encryption-based approach is that we can use a vector commitment scheme to handle multiple values in a single commitment, drastically improving the efficiency, as shown in Section 5.3.4. In Section 5.3.1, we show that using this approach, we can build a publicly tally-hiding system such that a single trustee can locally and thus efficiently compute a proof showing the correctness of the result.

5.3. Kryvos

Many verifiable e-voting systems, such as Ordinos (see Section 3.2), follow the concept of the prominent Helios system [AdMPQ09] (see Section 2.1). Helios is based on a $(t, n_{\mathsf{trustees}})$-threshold IND-CPA-secure additively homomorphic public-key encryption scheme, such as exponential ElGamal. Essentially, given the encrypted votes of the voters, the trustees homomorphically aggregate the encrypted votes to compute the publicly known encrypted tally $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ consisting of a sequence of ciphertexts containing the total number of votes for each choice component, i.e., there is one ciphertext per choice component. By aggregating votes, Helios breaks the link between voters and their votes, thereby achieving ballot privacy, and hides individual votes within the aggregated tally $c_{\mathsf{agg}}^{\mathsf{choice}}$. The trustees then decrypt $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$, i.e., all ciphertexts therein, in a distributed way and publish $c_{\mathsf{agg}}^{\mathsf{choice}}$ along with proofs of correct decryption, which reveals the aggregated tally $c_{\mathsf{agg}}^{\mathsf{choice}}$ and allows everyone to publicly compute the result of the election $f^{\mathrm{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$, e.g., the winner of the election. In systems that aim for full tally-hiding, i.e., where only $f^{\mathrm{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$, but not $c_{\mathsf{agg}}^{\mathsf{choice}}$ is revealed (see Chapter 3), trustees essentially perform certain forms of MPC on $E_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ in order to compute $f^{\mathrm{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$. Such heavy-weight MPC protocols take hours to evaluate elections (see Section 3.2.3), which is why in this work, we, for the first time, directly follow the standard approach of public tally-hiding, i.e., while trustees may learn $c_{\mathsf{agg}}^{\mathsf{choice}}$, everybody else should only learn $f^{\mathrm{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$.

In Section 5.3.1, introduce the Kryvos framework. Then, in Section 5.3.2, the security and privacy of the generic Kryvos framework are formally stated and proven. In Section 5.3.3, we present general-purpose proof systems, a primitive that we will use to instantiate the Kryvos framework. In Section 5.3.4 discusses various design choices and derive efficient realizations of core building blocks of our Kryvos instantiations. We present a detailed evaluation in Section 5.3.5. We provide the implementation in [HKK+23].

5.3.1. The Kryvos Framework

In this section, we present the Kryvos e-voting framework, the first verifiable e-voting system that directly follows the publicly tally-hiding paradigm. Kryvos is a generic framework supporting many voting methods and result functions.

The description of the Kryvos framework focuses on those parts of Kryvos that are at the core of the public tally-hiding property. Well-known and standard enhancements to security, in particular distributed implementations of bulletin boards (e.g., [CS14, KKL+18, HSB21]) and mechanisms to mitigate trust on the voting devices (e.g., [Ben07, GGP15]), are orthogonal to and fully compatible with Kryvos. Therefore, for simplicity of presentation, in the following, we identify voters and their voting devices. In particular, it is guaranteed that the ballots of honest voters contain the correct choice. Also, the modeling is such that ballots of honest voters are guaranteed to reach the bulletin board. These modeling choices and the security proofs can easily be lifted analogously to the modeling and security proofs of the Ordinos framework (Section 3.2.1).

5.3.1.1. Participants

The Kryvos protocol is run among a voting authority Auth, (human) voters $\mathsf{v}_1, \ldots, \mathsf{v}_{n_{\mathsf{voters}}}$, trustees $\mathsf{T}_1, \ldots, \mathsf{T}_{n_{\mathsf{trustees}}}$, and a bulletin board (BB). We assume that a mutually authenticated channel to BB exists for each party.

5.3.1.2. Voting Method and Election Result Function

The tuple $(\mathfrak{C} \subseteq \mathbb{F}_q^{n_{\mathsf{components}}}, f^{\mathrm{res}})$ consisting of a choice space (see Section 2.6) and an election result function (see Section 2.7) defines the abstract voting method for which we describe Kryvos; Section 5.3.5 presents specific instantiations.

5.3.1.3. Setup Phase

In this phase, all election parameters are fixed and posted on the BB by Auth: the list $\vec{id}$ of eligible voters, opening and closing times, the election ID id_{election}, and the voting method $(\mathfrak{C}, f^{\mathrm{res}})$. Additionally, Auth publishes a decomposition $n_{\mathsf{components}} = n_{\mathsf{tuples}} \cdot N$ that induces a decomposition on the choice space $\mathfrak{C} \subseteq (\mathbb{F}_q)^{n_{\mathsf{components}}} = (\mathbb{F}_q)^N \times \ldots \times (\mathbb{F}_q)^N$ yielding a splitting of a vote $c_i^{\mathsf{choice}} \in \mathfrak{C}$ into n_{tuples} tuples, each of size N. Intuitively, each part $(\mathbb{F}_q)^N$ is handled using a single general-purpose proof, leading to n_{tuples} many proofs showing the validity of a ballot. Therefore, this parameter allows for a trade-off of various benchmarks of the Kryvos instantiation: Using choice spaces with larger values of $n_{\mathsf{components}}$ might make a single general-purpose proof inefficient in terms of prove creation time or other benchmark metrics, depending on the concrete general-purpose proof system instantiation. Therefore, splitting the choice space into chunks such that independent proofs can evaluate chunks independently might allow for more efficient proofs of ballot validity. However, increasing the value of n_{tuples} leads to more proofs per ballot, and thus more proofs overall, possibly leading to more memory consumption, particularly on the BB. Furthermore, handling more commitments also increases the complexity of the proof showing the validity of the election result. In our instantiations in Section 5.3.5, we will always use $n_{\mathsf{tuples}} = 1$, except for IRV, where $n_{\mathsf{components}}$ is of exponential size in the number of candidates. We will discuss how this influences the Kryvos instantiation in Section 5.3.5.

A vote $c_i^{\mathsf{choice}} = (v_1^i, \ldots, v_{n_{\mathsf{tuples}}}^i)$ contains in v_1^i the entries for the first N choices, in v_2^i the next N choices and so on. If $n_{\mathsf{components}}$ is not divisible by N, we can artificially grow the choice space so that $n_{\mathsf{components}}$ is a multiple of N by appending zeros to a vote. This assignment is published by Auth on BB.

Furthermore, Auth creates and publishes the CRSs required for the general-purpose proofs on BB. Our system requires honestly generated CRSs for the Groth16 SNARKs. Several well-known mechanisms are practical with the parameters we need, such as distributed generation of the CRS by multiple parties, see Section 5.3.3. For simplicity and since this is an orthogonal issue, we here compute CRSs on a local computer. Each trustee T_k runs the key generation algorithm $\mathsf{KeyGen}_{\mathcal{E}}$ of an IND-CCA2-secure public-key encryption scheme $\mathcal{E} = (\mathsf{KeyGen}_{\mathcal{E}}, E_{\mathsf{pk}}, \mathsf{dec})$ to generate its public/private (encryption/decryption) key pair $(\mathsf{pk}_k, \mathsf{sk}_k)$ and posts (k, pk_k) on BB.

5.3.1.4. Voting Phase

Let $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}}) \in \mathfrak{C}$ be the vote of voter v_i. The voter creates full-threshold secret sharings of every component $c_{j,i}^{\mathsf{component}}$ of her vote to share $c_{j,i}^{\mathsf{component}}$ among the n_{trustees} trustees:

$$(c_{j,i,1}^{\mathsf{component}}, \ldots, c_{j,i,n_{\mathsf{trustees}}}^{\mathsf{component}}) \text{ with } \sum_{k=1}^{n_{\mathsf{trustees}}} c_{j,i,k}^{\mathsf{component}} \mod q = c_{j,i}^{\mathsf{component}}.$$

Then, for each trustee T_k, the voter decomposes $(c_{1,i,k}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i,k}^{\mathsf{component}})$ into chunks $(t_k^{i,1}, \ldots, t_k^{i,n_{\mathsf{tuples}}})$, where $n_{\mathsf{components}} = n_{\mathsf{tuples}} \cdot N$ is as defined by Auth in the setup phase (see above). Now, the voter creates a commitment to each tuple:

$$c_k^{i,l} \leftarrow \mathsf{com}(t_k^{i,l}, r_k^{i,l}).$$

Observe that since the commitment scheme is additively homomorphic, the commitment $c^{i,l} := \sum_{k=1}^{n_{\mathsf{trustees}}} c_k^{i,l}$ opens to the tuple $c_{l,i}^{\mathsf{component}} = (c_{1+(l-1)\cdot N,i}^{\mathsf{component}}, \ldots, c_{l\cdot N,i}^{\mathsf{component}})$, i.e., the votes for the choices belonging to the l-th factor in the decomposition of $(\mathbb{F}_q)^{n_{\mathsf{components}}}$ specified by Auth, using opening value $r^{i,l} := \sum_{k=1}^{n_{\mathsf{trustees}}} r_k^{i,l}$. One can obtain commitments on the original vote of v_i by combining all of the commitments on the full-threshold shares.

To guarantee the well-formedness of all commitments and the vote contained therein, the voter creates a proof $\pi_i^{\mathfrak{C}}$ proving that $(c^{i,1}, \cdots, c^{i,n_{\mathsf{tuples}}})$ commits to a vote $c_i^{\mathsf{choice}} \in \mathfrak{C}$.

For each trustee T_k, the voter uses T_k's public key pk_k to securely send the opening values for $(c_k^{i,1}, \ldots, c_k^{i,n_{\mathsf{tuples}}})$ to T_k:

$$e_k^i \leftarrow E_{\mathsf{pk}_k}((t_k^{i,1}, \ldots, t_k^{i,n_{\mathsf{tuples}}}), (r_k^{i,1}, \ldots, r_k^{i,n_{\mathsf{tuples}}})).$$

To provide ballot privacy, we assume that all plaintexts have a fixed length, which can easily be guaranteed here.

To complete the voting process, v_i submits her ballot $\mathsf{b}_i = (i, (c_k^{i,l})_{l,k}, \pi_i^{\mathfrak{C}}, (e_k^i)_k)$ to the bulletin board. BB then verifies that i) the voter is eligible to vote, ii) has not submitted a valid ballot before, iii) the voter's SNARK proof is valid, and iv) no voter has previously submitted a vector containing any of the ciphertexts in $(e_k^i)_k$. This so-called *ballot weeding* process prevents malicious voters from submitting ballots that are related to the ballots from honest voters, which would break privacy. In particular, due to the CCA2-secure encryption, which must contain the correct randomness for the commitments, the only way to submit a valid related ballot is by creating a new ballot that re-uses some of the (unmodified) ciphertexts from the honest ballots. If at least one trustee is honest (which we always assume for privacy), the ballot weeding procedure prevents ballot weeding. If all checks succeed, the BB adds b_i to the list of ballots $\vec{\mathsf{b}}$ and publicly updates $\vec{\mathsf{b}}$.

The trustees must prepare the list $\vec{\mathsf{b}}$ for the tallying phase. Each trustee T_k decrypts every e_k^i posted on BB:

$$((\tilde{t}_k^{i,1}, \ldots, \tilde{t}_k^{i,n_{\mathsf{tuples}}}), (\tilde{r}_k^{i,1}, \ldots, \tilde{r}_k^{i,n_{\mathsf{tuples}}})) \leftarrow \mathsf{dec}_{\mathsf{sk}_k}(e_k^i).$$

Then T_k checks whether each pair $(\tilde{t}_k^{i,l}, \tilde{r}_k^{i,l})$ is a valid opening for $c_k^{i,l}$. If this is not the case, then T_k publishes a NIZKP π^{dec} of correct decryption of e_k^i on BB so that one can verify that e_k^i is invalid; as a consequence, the corresponding ballot will not be counted. For example, one can combine the IND-CCA2-secure PKE by Cramer-Shoup [FHR21] with the NIZKP by Camenisch-Shoup [CS03]. All of the following steps, including the tallying phase, are performed only for those ballots that have passed this check; for simplicity of presentation, we assume that all voters have submitted a valid ballot.

5.3.1.5. Tallying Phase

Everyone can homomorphically aggregate the public commitments on BB by computing

$$c^{\mathsf{agg},l} \leftarrow \sum_{k=1}^{n_{\mathsf{trustees}}} \sum_{i=1}^{n_{\mathsf{voters}}} c_k^{i,l}$$

for each $1 \leq l \leq n_{\mathsf{tuples}}$. In parallel, the trustees homomorphically aggregate the corresponding opening values. First, each trustee T_k internally computes for each $1 \leq l \leq n_{\mathsf{tuples}}$

$$t_k^{\mathsf{agg},l} \leftarrow \sum_{i=1}^{n_{\mathsf{voters}}} t_k^{i,l}; \quad r_k^{\mathsf{agg},l} \leftarrow \sum_{i=1}^{n_{\mathsf{voters}}} r_k^{i,l},$$

where the $t_k^{i,l}$'s and $r_k^{i,l}$'s are the opening values decrypted above. Next, each trustee T_k shares $(t_k^{\mathsf{agg},l})_l$ and $(r_k^{\mathsf{agg},l})_l$ with the other trustees so that for each $1 \leq l \leq n_{\mathsf{tuples}}$, the trustees can internally compute

$$t^{\mathsf{agg},l} \leftarrow \sum_{k=1}^{n_{\mathsf{trustees}}} t_k^{\mathsf{agg},l}; \quad r^{\mathsf{agg},l} \leftarrow \sum_{k=1}^{n_{\mathsf{trustees}}} r_k^{\mathsf{agg},l}.$$

The trustees compute the (aggregated) tally of all votes $c_{\mathsf{agg}}^{\mathsf{choice}} = (c_{1,\mathsf{agg}}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},\mathsf{agg}}^{\mathsf{component}})$, where $c_{j,\mathsf{agg}}^{\mathsf{component}}$ is the total number of votes for (choice) component $c_j^{\mathsf{component}}$.

A designated trustee then performs the final step. First, this trustee computes the election result $\mathsf{elecres} \leftarrow f^{\mathrm{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$. She then computes a proof $\pi^{f^{\mathrm{res}}}$ using $\Pi_{f^{\mathrm{res}}}$ on public inputs $c^{\mathsf{agg},1}, \ldots, c^{\mathsf{agg},n_{\mathsf{tuples}}}$ and $\mathsf{elecres}$ that proves knowledge of $c_{\mathsf{agg}}^{\mathsf{choice}}$ (as well as knowledge of randomness) such that $c^{\mathsf{agg},1}, \ldots, c^{\mathsf{agg},n_{\mathsf{tuples}}}$ is a list of commitments on $c_{\mathsf{agg}}^{\mathsf{choice}}$ and $\mathsf{elecres}$ is the output of $f^{\mathrm{res}}(c_{\mathsf{agg}}^{\mathsf{choice}})$. The trustee publishes $\mathsf{elecres}$ and the proof BB. We will define the formal relation $\mathcal{R}_f^{\mathrm{res}}$ of this ZKP in Section 5.3.5.

5.3.1.6. Public Verification Phase

In this phase, every participant, including the voters or external observers, can verify the correctness of the tallying procedure, particularly all ZKPs. In particular, external observers can re-compute the homomorphic aggregations of commitments without knowing any openings.

5.3.2. Security of Kryvos

In this section, we formally show that Kryvos enjoys verifiability and privacy (including publicly tally-hiding) properties.

5.3.2.1. Computational Model of Kryvos

We start by formally modeling Kryvos using the general computational framework presented in Section 2.2 that we use both for analyzing the verifiability and privacy of Kryvos.

We model Kryvos as a protocol $\mathsf{P}_{\mathsf{Kryvos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, \mathfrak{C}, f^{\mathrm{res}})$ following the description from Section 5.3.1. Recall from that section that we denote the number of voters by n_{voters}, the number of trustees by n_{trustees}, and the voting method by $(\mathfrak{C}, f^{\mathrm{res}})$. By μ, we denote a probability distribution over $\mathfrak{C}$ according to which each honest voter makes her choice. Note that by this, we model that the adversary knows this distribution. This choice is called the *actual choice* of the voter. Besides the abovementioned parties, Kryvos contains a BB. In our model of Kryvos, the voting authority Auth is part of an additional agent, the scheduler S. Besides playing the role of the authority, S schedules all other agents in a run according to the protocol phases. In particular, we assume that the voting authority Auth (more generally, the scheduler S) and the BB are honest, i.e., the adversary does not corrupt these parties.

The judging procedure of Kryvos is very simple. The judge reads all data from the bulletin board BB and accepts the run if and only if all NIZKPs on BB are valid (and only eligible voters voted, at most once): the voters' NIZKPs $\pi_i^{\mathfrak{C}}$ to prove ballot validity, the trustees' NIZKPs $\pi^{f^{\text{res}}}$ to prove the correctness of the final result, and (if any) the trustees' NIZKP π^{dec} to prove that a voters' ciphertext does not decrypt to a valid opening of her commitments.

5.3.2.2. Verifiability

In this section, we establish the level of verifiability provided by Kryvos for the goal $\gamma(k, \varphi)$ presented in Definition 2.1. We prove the verifiability result for Kryvos under the following assumptions:

(V1) The encryption scheme is correct, the commitment scheme is homomorphic and computationally binding, and all NIZKPs are (computationally) sound.

(V2) The scheduler S, the judge J, and the BB are honest: $\varphi = \mathsf{hon}(\mathsf{S}) \wedge \mathsf{hon}(\mathsf{J}) \wedge \mathsf{hon}(\mathsf{BB})$.

Note that the adversary may control an arbitrary number of voters and trustees.

Theorem 5.1 (Verifiability of Kryvos)**.** *Under the assumptions (V1) and (V2) states above and the mentioned judging procedure run by the judge* J*,* $\mathsf{P}_{\mathsf{Kryvos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, \mathfrak{C}, f^{\text{res}})$ *is* $(\gamma(\infty, \varphi), 0)$*-verifiable, w.r.t. the judge* J*.*

We note that this theorem differs from Theorem 3.3. As explained in Section 5.3.1, our modeling described, is such that votes of honest voters are cast as intended and reach the bulletin board. Therefore, the theorem holds even though voters do not perform any verification. Also, instead of considering k-risk-avoiding adversaries for some $k < \infty$, we can consider arbitrary probabilistic polynomial-time adversaries, i.e., $k = \infty$.

If we assume the presence of malicious voting devices or that ballots might be dropped before reaching the bulletin board, we can reintroduce the security mechanisms considered also for Ordinos (verification by voters with probabilities p_{verify} and p_{audit}). In this case, analogously to Ordinos, we obtain $(\gamma(k, \varphi), \delta_k(p_{\mathsf{verify}}, p_{\mathsf{audit}}))$-verifiability for Kryvos.

We provide our formal proof of the verifiability theorem in Appendix B.1. This result mainly uses the soundness of the NIZKPs/SNARKs employed in Kryvos. The underlying relations of these proofs yield a *global* relation, which effectively ensures the goal Φ_k.

5.3.2.3. Accountability

The issue regarding Kryvos's accountability property lies in the internal communication between the trustees. During the tallying phase, the trustees internally open the aggregated tally by sending their openings to the other trustees. However, it is not possible to resolve complaints cast in this phase. If trustee i publicly raises a complaint stating "Trustee j did not open her commitments," the public cannot judge what happened:

1. Trustee j did not open her commitments, or

2. Trustee i falsely accuses trustee j.

The issue arises by hiding the log of the internal communication between the trustees. Nevertheless, we cannot publish the communication log after a complaint is raised for privacy reasons.

5.3.2.4. Privacy

We define the set of admissible adversaries for Kryvos as follows. The process $\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}}$ of $\mathsf{P}_{\mathsf{Kryvos}}$, the set $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}})$ of admissible adversaries for $\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}}$ is defined as follows. An adversary A belongs to $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}})$ if (and only if) it satisfies the following conditions: (i) $\mathfrak{p}_{\mathsf{A}}$ is k-risk-avoiding for $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}}$, (ii) and the probability that $\mathfrak{p}_{\mathsf{A}}$ corrupts more than $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters in a run of $\mathfrak{P}_{\mathsf{P}_{\mathsf{Ordinos}}} \| \mathfrak{p}_{\mathsf{A}}$ is negligible.

To analyze the publicly tally-hiding property of Kryvos, we make the following assumptions about the primitives we use (see also Section 5.3.1):

(P1) The public-key encryption scheme is IND-CCA2-secure, the SNARKs and the NIZKPs are perfectly zero-knowledge, and the commitment scheme is perfectly hiding.

(P2) An adversary $\mathfrak{p}_{\mathsf{A}}$ does neither corrupt the scheduler S nor the BB, and at least $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters are honest.

(P3a) An adversary $\mathfrak{p}_{\mathsf{A}}$ does not corrupt any trustee.

(P3b) An adversary $\mathfrak{p}_{\mathsf{A}}$ does corrupt at most $t-1$ many trustees.

Theorem 5.2 (Public Privacy)**.** *Under the assumptions (P1), (P2), (P3a), and the mapping* $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}})$ *stated above, the voting protocol* $\mathsf{P}_{\mathsf{Kryvos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, \mathfrak{C}, f^{\mathrm{res}})$ *of* Kryvos *achieves* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ *privacy.*

Theorem 5.3 (Internal Privacy)**.** *Under the assumptions (P1), (P2), (P3b), and the mapping* $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}})$ *stated above, the voting protocol* $\mathsf{P}_{\mathsf{Kryvos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, \mathfrak{C}, f^{\mathrm{res}})$ *of* Kryvos *achieves* $\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$ *privacy.*

Theorem 5.4 (Publicly Tally-Hiding)**.** *Under the assumptions (P1), (P2), (P3a), (P3b), and the mapping* $\mathfrak{A}(\mathfrak{P}_{\mathsf{P}_{\mathsf{Kryvos}}})$ *stated above, the voting protocol* $\mathsf{P}_{\mathsf{Kryvos}}(n_{\mathsf{voters}}, n_{\mathsf{trustees}}, \mu, \mathfrak{C}, f^{\mathrm{res}})$ *of* Kryvos *achieves* $(\delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\mathrm{res}}), \delta^{ideal}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}}))$*-publicly tally-hiding w.r.t.* (T, t).

The formal proofs of Theorems 5.2 to 5.4 are provided in Appendix B.2. The proofs are based on two sequences of games, one for internal and one for public privacy.

Theorem 5.4 essentially states that the public privacy level δ_p of Kryvos is the ideal one for $(\mathfrak{C}, f^{\mathrm{res}})$, and its internal privacy level δ_i is the ideal one for $(\mathfrak{C}, f_{\mathsf{AggTally}})$ where f_{AggTally} returns the number of votes for each choice/candidate. Figure 3.7 illustrates that for IRV, Kryvos dramatically improves public privacy compared to non-tally-hiding systems (for which $\delta_i = \delta_p \geq \delta^{\mathrm{ideal}}(\mathfrak{C}, f_{\mathsf{AggTally}})$).

5.3.3. General-Purpose Proof Systems

An integral part of the Kryvos e-voting scheme is the general-purpose proof system (GPPS) that show (i) validity of ballots, and (ii) validity of the election result function. Instead of using specialized ZKPs that support exactly one such relation for a specific election result function, using general-purpose proof systems allows for high flexibility of the Kryvos framework to support different choice spaces and election result functions. This section will explore the features of GPPSs and choose an appropriate instantiation for Kryvos.

A GPPS takes as input an arbitrary indicator function $f_{\mathcal{R}} : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}$ for some binary relation $\mathcal{R}$, such that $f_{\mathcal{R}}(x,w) = 1$ if (and only if) $(x,w) \in \mathcal{R}$ for public statement x and secret knowledge/*witness* w. It then allows for computing a zero-knowledge proof that proves knowledge of w such that $f_{\mathcal{R}}(x,w) = 1$. In our setting, to prove validity of the election result accordingly to f^{res}, a trustee can use $x = (\mathsf{Com}_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}}), y)$ as public input along with a suitable witness (which, among others, contains some tally $c_{\mathsf{agg},w}^{\mathsf{choice}}$) and the function $f_{\mathcal{R}}$: $f_{\mathcal{R}}(x,w) = 1$ if (and only if) $c_{\mathsf{agg},w}^{\mathsf{choice}}$ corresponds to the plaintext in $\mathsf{Com}_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$ and $y = f^{\text{res}}(c_{\mathsf{agg},w}^{\mathsf{choice}})$. Usually, the GPPS represents the indicator function $f_{\mathcal{R}}$ as an arithmetic circuit, and the complexity of the proof systems depends on the number of gates of this circuit. Regarding the election result functions, relations/circuits for our setting of publicly tally-hiding e-voting consist of two parts:

1. Showing correctness of cryptographic primitives: $c_{\mathsf{agg},w}^{\mathsf{choice}}$ corresponds to the plaintext in $\mathsf{Com}_{\mathsf{pk}}^{\mathsf{Vector}}(c_{\mathsf{agg}}^{\mathsf{choice}})$.
2. Evaluating the tally: $y = f^{\text{res}}(c_{\mathsf{agg},w}^{\mathsf{choice}})$.

Therefore, the concrete GPPS must efficiently support both parts to obtain an efficient publicly tally-hiding e-voting system. The GPPS must handle complex result functions even for more significant numbers of candidates/choices, which, among others, requires reasonably low proof creation and verification times. However, as we will show in Section 5.3.5, showing the validity of an opening of a commitment is far more demanding than evaluating the election result. Performing cryptographic operations inside GPPSs quickly reaches the limitations of such systems. Moreover, the GPPS for our e-voting system must satisfy several requirements to be applicable, see Section 5.2.1. Therefore, we use highly efficient GPPSs, namely succinct non-interactive arguments of knowledge, introduced next to obtain efficient realizations.

5.3.3.1. SNARKs

In order to efficiently prove the opening of a commitment and the correct evaluation of an election result function, we need general proof systems that are efficient and scale well, even for large circuits. Zero-knowledge succinct non-interactive argument of knowledge [GMO16, Gro16, AHIV17, BBC+18, BBHR18, BBB+18, BCR+19, ESLL19, ESS+19, GWC19, MBKM19, AC20, ALS20, BFH+20, BLNS20, COS20, ENS20, Set20, ACK21, AL21, BCS21, ISW21, LNPS21, ACC+22,

ACL+22, CHJ+22, Eag22, LNP22, BS23, ABST23, CBBZ23, CGG+23, CGI+23, GLS+23, Zha23] (SNARKs, for short, where we implicitly assume the zero-knowledge property) are highly efficient general-purpose proof systems and these currently fit these requirements best. Essentially, they are general-purpose proof systems with the additional property of succinctness. In general, succinctness states that the proof size and the verification time are sublinear in the size of the circuit (that encodes the indicator function $f_{\mathcal{R}}$). In order to obtain succinctness, SNARKs have the trade-off that they loosen the notion of soundness. That is, instead of achieving perfect soundness, they achieve computational soundness.

In the following, following [Gro16], we define zero-knowledge succinct non-interactive arguments of knowledge (SNARKs).

Let $\mathcal{R}$ be a relation generator given a security parameter η in unary, returns a polynomial time decidable binary relation $\mathcal{R}$. For pairs $(x, w) \in \mathcal{R}$, we call x the (public) statement and w the (secret) witness. We define $\mathcal{R}_\eta$ as the set of possible relations $\mathcal{R}$ that the relation generator may output given 1^η. In the following, we will assume that we can deduce η from the description of $\mathcal{R}$. The relation generator may also output some side information, an auxiliary input z, given to the adversary. An *efficient prover publicly verifiable non-interactive argument* for $\mathcal{R}$ is a tuple of probabilistic polynomial time algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ such that:

- $(\mathsf{EKCRS}, \mathsf{VKCRS}, \tau) \leftarrow \mathsf{Setup}(\mathcal{R})$: The setup produces two common reference strings (CRSs): EKCRS (*evaluation key* CRS, needed to create proofs) and VKCRS (*verification key* CRS, needed to verify proofs), and a simulation trapdoor τ for the relation $\mathcal{R}$. Typically, VKCRS is included in EKCRS.
- $\pi \leftarrow \mathsf{Prove}(\mathcal{R}, \mathsf{EKCRS}, x, w)$: The prover algorithm takes as input a common reference string (CRS) EKCRS and $(x, w) \in \mathcal{R}$ and returns an argument π.
- $0/1 \leftarrow \mathsf{Verify}(\mathcal{R}, \mathsf{VKCRS}, x, \pi)$: The verification algorithm takes as input a common reference string (CRS) VKCRS, a statement x, and an argument π, and returns 0 (reject) or 1 (accept).
- $\pi \leftarrow \mathsf{Sim}(\mathcal{R}, \tau, x)$: The simulator takes as input a simulation trapdoor τ and statement x, and returns an argument π.

Definition 5.2. *The tuple* $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ *is a perfect non-interactive zero-knowledge argument of knowledge for* $\mathcal{R}$ *if it has perfect completeness, perfect zero-knowledge and computational knowledge soundness as described below.*

In what follows, we describe the main idea of the respective properties and refer to [Gro16] for the formal definitions:

- Perfect zero-knowledge: An argument is *zero-knowledge* if it does not leak any information besides the statement's truth.
- Computational soundness: $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ is *sound* if it is impossible to prove a false statement.

- Computational knowledge soundness: The tuple $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify}, \mathsf{Sim})$ is an argument of knowledge if an extractor can compute a witness whenever the adversary produces a valid argument.

5.3.3.2. Selecting a SNARK for Kryvos

Our setting of publicly tally-hiding e-voting poses several requirements that a SNARK must fit, see Section 5.2.1. That is, the proof system should be as efficient as possible in every aspect: The size of the common reference strings (CRSs) should be as small as possible, proof generation should be as fast as possible, the proofs should be as small as possible, and verification should be as fast as possible. Furthermore, the SNARK should be executable even on devices with low computational power so voters can create and verify proofs.

There are many possible SNARKs for instantiating the zero-knowledge proofs for Kryvos. Groth proposed a SNARK [Gro16] (called *Groth16 SNARK* in the following) that is state-of-the-art and possesses the best benchmarks in comparison with other SNARKs: It posses constant proof size and verification time (independent of the circuit) while maintaining reasonable efficient prover times. We will discuss this SNARK next. Our construction is based on arithmetic circuits and is not limited to the Groth16 SNARK. We can instantiate Kryvos with every proof system based on arithmetic circuits (such as, e.g., [Gro16, AHIV17, BBC+18, BBB+18, BCR+19, COS20, Set20, CHJ+22, Eag22, BS23, GLS+23]).

Furthermore, we will instantiate the Kryvos framework with Pedersen commitments (see following sections). We emphasize that the (results for the) efficient combination of algorithms for computing Pedersen commitments will also carry over to other SNARKs as long as the base field of the elliptic curve for the Pedersen commitments is compatible with the base field of the SNARK. While the Groth16 SNARK is compatible with our construction, it might require some work for other SNARKs.

5.3.3.3. The Groth16 SNARK

The Groth16 SNARK [Gro16] is a highly efficient state-of-the-art SNARK that offers constant proof size with constant verification time (independently of the function $f_{\mathcal{R}}$) while achieving a relatively fast polynomial proving time and thus scaling well even for highly complex functions.

Simplified, the Groth16 SNARK consists of three algorithms: $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$. The $\mathsf{Setup}(f_{\mathcal{R}})$ algorithm generates two common reference strings (CRSs), EKCRS (*evaluation key* CRS, needed to create proofs) and VKCRS (*verification key* CRS, needed to verify proofs) that depend on $f_{\mathcal{R}}$. The setup creates an instance of Groth16 specific to the function $f_{\mathcal{R}}$. Anyone can then use EKCRS to create a proof $\pi \xleftarrow{\$} \mathsf{Prove}(\mathsf{EKCRS}, x, w)$ with the abovementioned properties. One can use $\mathsf{Verify}(\mathsf{VKCRS}, x, \pi)$ to verify the proof, which requires only a very small VKCRS. The Groth16 SNARK operates on bilinear groups (defined in Appendix A.3) whose order depends on the size of the input values. The currently most efficient implementation [sci17], which we

use, uses bilinear groups with size up to 256 bits and supports operations in $\mathbb{F}_p^*$ with p of size up to 255 bits.

A Groth16 SNARK uses the language of quadratic arithmetic programs to specify the underlying relation $\mathcal{R}$ and indicator function $f_{\mathcal{R}}$, which we will define next.

5.3.3.4. Quadratic Arithmetic Programs (QAPs)

Consider an arithmetic circuit of addition and multiplication gates over a finite field $\mathbb{F}$. We may designate some of the input/output wires as specifying a statement and use the rest of the wires in the circuit to define a witness. The combination of statement and witness gives us a binary relation $\mathcal{R}$ consisting of statement wires and witness wires that satisfy the arithmetic circuit, i.e., make it consistent with the designated input/output wires. Generalizing arithmetic circuits, we may be interested in relations described by constraints over a set of variables (in this work, we will use the terms wire and variable interchangeable). Some variables correspond to the statement; the remaining variables correspond to the witness. The relation consists of statements and witnesses that satisfy all the constraints.

Definition 5.3 (Constraint)**.** *Let $(a_i)_{i=0}^m \in \mathbb{F}^{m+1}$ be a set of variables with $a_0 = 1$. A constraint consists of constants $(u_i, v_i, w_i)_{i=0}^m \in (\mathbb{F}, \mathbb{F}, \mathbb{F})^{m+1}$ and we define that the constraint is satisfied if the following holds:*

$$\sum_{i=0}^{m} u_i a_i \cdot \sum_{i=0}^{m} v_i a_i = \sum_{i=0}^{m} w_i a_i.$$

Addition and multiplication gates are special cases of such constraints, so such systems of arithmetic constraints generalize arithmetic circuits. A multiplication gate can for instance be described as $a_i \cdot a_j = a_k$ (using $u_i = 1, v_j = 1$ and w_k and setting the remaining constants for this gate to 0). We can handle addition gates for free, i.e., if $a_i + a_j = a_k$ and a_k is multiplied by a_l, we may write $(a_i + a_j) \cdot a_l$ and skip the calculation of a_k.

As described by Parno et al. [PHGR13, Gro16], we can reformulate the set of arithmetic constraints as a quadratic arithmetic program assuming $\mathbb{F}$ is large enough. Given n constraints we pick arbitrary distinct $r_1, \ldots, r_n \in \mathbb{F}$ and define $t(x) = \Pi_{q=1}^{n}(x - r_q)$. Furthermore, let $p_i^u(x)$, $p_i^v(x)$, $p_i^w(x)$, be degree $n-1$ polynomials such that $p_i^u(x) = u_{i,x}$, $p_i^v(x) = v_{i,x}$, $p_i^w(x) = v_{i,x}$ for $x \in \{1, \ldots, m\}$ where $u_{i,x}$, $v_{i,x}$, $w_{i,x}$ are the constants u_i, v_i, w_i of the x-th constraint.

Definition 5.4 (QAP [Gro16])**.** *A quadratic arithmetic program* QAP *over the field $\mathbb{F}$ contains three sets of polynomials $(p_i^u(x), p_i^v(x), p_i^w(x))_{i=0}^m$ and a target polynomial $t(x)$ over $\mathbb{F}[x]$ such that $\deg(p_i^u(x)), \deg(p_i^v(x)), \deg(p_i^w(x)) < d := \deg(t(x))$ for all $i = 0, \ldots, m$.*

We define that QAP *accepts an input $a_1, \ldots, a_l \in \mathbb{F}$ if and only if there exist $(a_i)_{i=l+1}^m$ satisfying (for $a_0 = 1$):*

$$\sum_{i=0}^{m} p_i^u(x)a_i \cdot \sum_{i=0}^{m} p_i^v(x)a_i \equiv \sum_{i=0}^{m} p_i^w(x)a_i \mod t(x)$$

5.3.3.5. Secure CRS Generation for the Groth16 SNARK

If the CRSs for the Groth16 SNARKs used in Kryvos are not generated honestly by the voting authority Auth but were instead generated maliciously (*subverted*), then the soundness and the zero-knowledge properties of the SNARKs break down.

We can mitigate this trust assumption using standard techniques: To allow for verifying that a CRS provides the zero-knowledge property, one only has to add some additional elements to the CRS during its generation, as detailed in [BFS16, ALSZ21]. We note that the prover does not need to download these additional elements to compute a proof, i.e., this mechanism does not add to the overall size that a voter needs to download to submit a ballot. To additionally retain the soundness property, one can let the trustees use MPC to generate the CRS in a distributed fashion [BCG$^+$15a, BGM17, ABL$^+$19]. Under the assumption that at least one trustee is honest, which we already presume for privacy, the resulting common reference string (CRS) provides soundness.

Distributed CRS generation is indeed practical for the parameters used in our Kryvos instantiations presented in Section 5.3.5: The benchmarks of Bowe et al. [BGM17] for a system with 2^{21} constraints show that generating and checking the CRS took approximately an hour. The SNARKs used in our system are similar to those benchmarked in [BGM17] (in particular, we use approximately 5 million constraints). Therefore, we estimate our case to be in the same order of magnitude. We can generate the common reference strings (CRSs) before the election, and the generation must be performed only once for each voting method; distributed common reference string (CRS) generation is a viable option.

Alternatively, in cases where trusting a hardware manufacturer is an option (e.g., small-stake elections such as boardroom voting), one can use trusted execution environments (TEEs) such as [IBM20] or [AMD20] to generate a CRS honestly. We must select the TEE suitably, giving it access to true randomness and sufficient memory.

5.3.3.6. Benchmarks of the Groth16 SNARK

The overall performance (runtime, bandwidth, memory overhead) of a Groth16 SNARK depends on the number of constraints and, hence, the size of the arithmetic circuit. Furthermore, the number of public wires influences the time to verify proofs. However, in practical use cases, such as our e-voting scenario, the number of public wires is deficient concerning the total number of wires, such that this dependency does not influence the overall runtime of the Groth16 proof system. Nevertheless, we present benchmarks of circuits that almost consist of public wires only for completeness.

We created the benchmarks using the libsnark library [sci17], which uses a Barreto-Naehrig curve of 128 bits security as the underlying field of the Groth16 SNARK. Furthermore, we run the experiments on an ESPRIMO Q957 (64 bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM).

In Figure 5.2, we present various Groth16 SNARK benchmarks, where the circuits consist of one constraint per wire. We obtained the blue benchmarks using circuits that consist of precisely one public wire, and the remaining wires are all secret. The circuits of the red benchmarks consist of public wires only. In Figure 5.2a, we benchmark Setup in seconds, i.e., the time it takes to create the two common reference strings (CRSs) concerning the number of constraints. It takes slightly longer to perform the setup using more public wires. In Figure 5.2b, we present the size of the evaluation key common reference string (CRS) EKCRS in megabytes, and in Figure 5.2c the size of the verification key common reference string (CRS) VKCRS in kilobyte and megabyte in Figure 5.2d. For each wire, we add a corresponding group element to one of the common reference strings (CRSs): VKCRS contains the group element of a public wire. In contrast, EKCRS contains the group element of a secret wire. Therefore, the size of the VKCRS increases with rising numbers of public wires, while the size of the EKCRS slightly increases with rising numbers of secret wires. Due to its size without these elements, this does not affect the EKCRS as much as the VKCRS. The VKCRS becomes of the size of 1 MB with $6,278$ public wires.

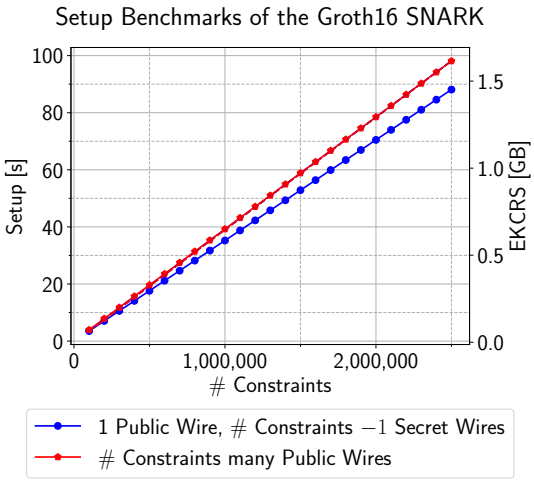
Furthermore, in Figure 5.2e, we benchmark Prove, the time it takes to generate a proof. The ratio between public and secret wires does not influence the proof generation time. In contrast, the time of the verification Verify of a proof, presented in Figure 5.2f, is directly influenced by the number of public wires because the verifier has to perform one operation per group element of the public wires.

The size of a Groth16 proof consists of three group elements independent of the relation of the proof. In our experiments, all of our proofs require 200 bytes.

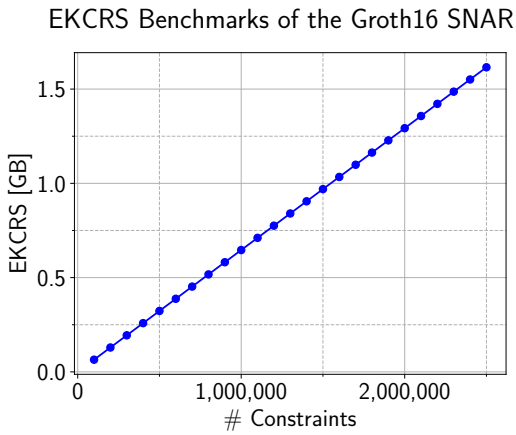
We remark that all of our circuits used in the Kryvos e-voting system consist of a meager number of public wires concerning the total number of wires, which is why our benchmarks in the following will be close to the blue benchmarks of Figure 5.2.

5.3.4. Designing Efficient Circuits for QAPs

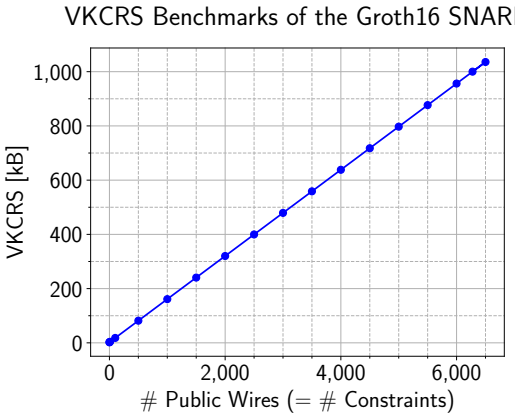
As shown in the previous section, the main factor determining the performance of the Groth16 SNARK is the number of constraints of the circuit relation. Thus, a crucial and labor-intensive step for obtaining an efficient SNARK is to optimize the implementation via constraints. While one can specify an arithmetic circuit defined in a suitable language and then automatically compute a representation via constraints, such a generic conversion results in needlessly large numbers of constraints and can lead to impractical SNARKs. Obtaining efficient SNARK proofs is crucial for our voting system. Therefore, we manually implemented and optimized our circuit via constraints. Our goal is to minimize the number of constraints in our circuits.



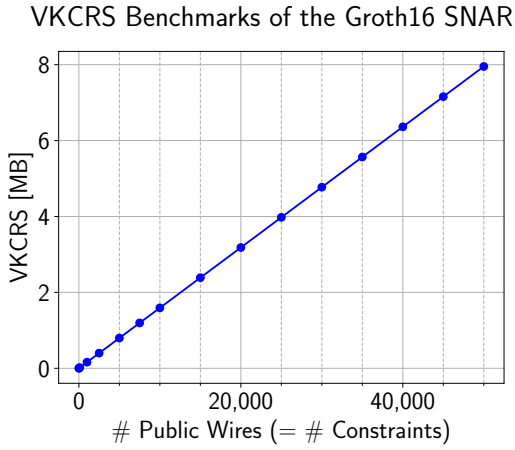
(a) Setup Benchmarks of the Groth16 SNARK.



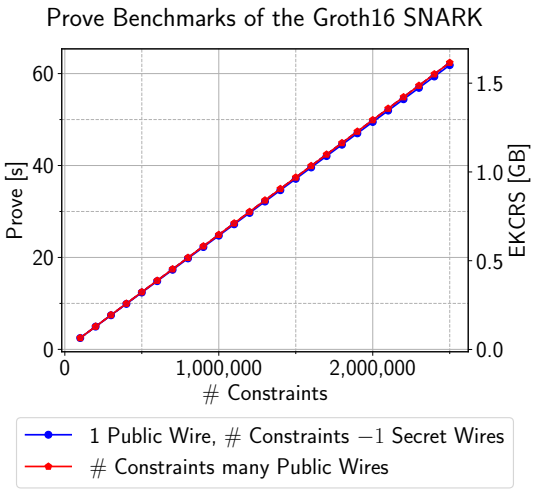
(b) EKCRS Benchmarks of the Groth16 SNARK.



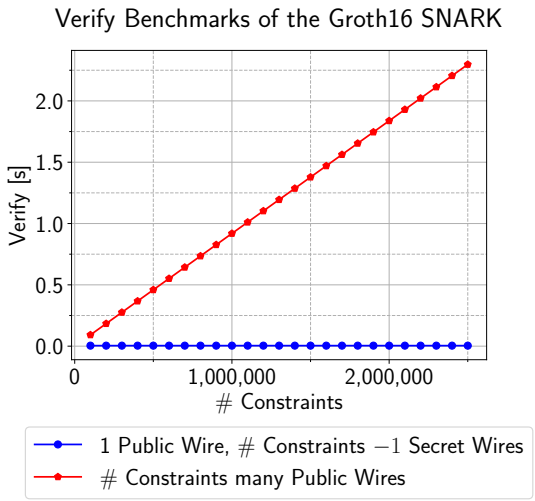
(c) VKCRS Benchmarks of the Groth16 SNARK with size up to 1 MB.



(d) VKCRS Benchmarks of the Groth16 SNARK for larger sizes.



(e) Prove Benchmarks of the Groth16 SNARK.



(f) Verify Benchmarks of the Groth16 SNARK.

Figure 5.2.: Benchmarks of the Groth16 SNARK.

A core concept of creating efficient circuits and creating efficient QAP constraints is verification of the computation. In the context of MPC (see Section 3.2), no party learns the result, and therefore, we need to perform the computation that leads to the result of the MPC protocol via MPC. However, in many cases, verifying a given result is much more efficient than computing it. For example, in the case of exponential ElGamal, one needs to brute force the final decryption, i.e., in order to obtain m from g^m, one needs to try all possible plaintext values x and check whether $g^x = g^m$ (and thus $x = m$). In the case of MPC, one would need to try all possible values x and store the one that satisfies the condition.

However, in the case of zero-knowledge proofs, we can follow a different path of constructing circuits: The prover knows the witness (or can compute it by herself), and thus, we need to verify the secret input of the prover. For example, in the case of exponential ElGamal, the prover inputs the correct x (which she might have computed by herself via brute force in advance). Then, the circuit verifies that the value of this wire satisfies the needed condition. In contrast to the computation, verifying a solution is more efficient in many use cases. Thus, our circuits generally aim at verification instead of computation.

5.3.4.1. Verification vs. Assertion

We can divide the concept of the verification of the witness into two sub-concepts, which we call *verification* and *assertion*. With *assertion*, we assert that some condition holds, which means that the circuit cannot be satisfied without the condition. We present two examples for this:

1. Consider a circuit $\mathsf{Circ}^{\mathsf{AssertEqual}}$ that takes as input two wires a_1, a_2 and enforces that $a_1 = a_2$, i.e., the circuit can only be satisfied if (and only if) $a_1 = a_2$. We can construct such a circuit with the constraint

$$(0 \cdot a_0 + 1 \cdot a_1 + 0 \cdot a_2) \cdot (1 \cdot a_0 + 0 \cdot a_1 + 0 \cdot a_2) = 0 \cdot a_0 + 0 \cdot a_1 + 1 \cdot a_2.$$

   This constraint sets $u_1, v_0, w_2 = 1$ and $u_0, u_2, v_1, v_2, w_0 w_1 = 0$. Recall that a_0 is the public wire with constant value 1. It is impossible to find an assignment of a_1, a_2 with $a_1 \neq a_2$ that satisfies this circuit.

2. The circuit $\mathsf{Circ}^{\mathsf{AssertBit}}$ takes as input one wire a_1 and enforces that $a_1 \in \{0, 1\}$, i.e., the circuit can only be satisfied if (and only if) $a_1 = 0$ or $a_1 = 1$. We can construct such a circuit with the constraint

$$(0 \cdot a_0 + 1 \cdot a_1) \cdot (1 \cdot a_0 + (-1) \cdot a_1) = 0 \cdot a_0 + 0 \cdot a_1.$$

   This constraint sets $u_1, v_0 = 1$, $v_1 = -1$, $u_0 w_0 w_1 = 0$, and reads $a_1 \cdot (1 - a_1) = 0$. Since $\mathbb{F}$ is a field, it has no nonzero divisors. Therefore, the constraint can only be satisfied if $a_1 = 0$ (and thus $a_1 = 0$), or $1 - a_1 = 0$ (and thus $a_1 = 1$), leading to $a_1 \in \{0, 1\}$.

While assertions typically lead to circuits with the lowest number of constraints, we often need to loosen the circuit to satisfy it without holding the condition. Consider a circuit with two input wires a_1, a_2 that is satisfied if $a_1 = a_2$ or $a_1 \in \{0, 1\}$.We can construct this circuit using $\mathsf{Circ}^{\mathsf{AssertEqual}}$ and $\mathsf{Circ}^{\mathsf{AssertBit}}$ as sub-circuits. However, if only one of the conditions holds, the circuit cannot be satisfied due to the assertion structure of the sub-circuits. Therefore, we realize such circuits via the concept of *verification*: We verify whether a condition holds and store the result in an indicator wire a_j, which is set to 1 if the condition holds and 0 otherwise. With that, we can compute two indicator wires, a_3 for $\mathsf{Circ}^{\mathsf{VerifyEqual}}$, and a_4 for $\mathsf{Circ}^{\mathsf{VerifyBit}}$, and then compute a logical or of these two wires. However, circuits that verify typically need more constraints than their assertion counterparts. In the following, we will construct $\mathsf{Circ}^{\mathsf{AssertEqual}}$ and $\mathsf{Circ}^{\mathsf{AssertBit}}$, which need more constraints than $\mathsf{Circ}^{\mathsf{AssertEqual}}$ and $\mathsf{Circ}^{\mathsf{AssertBit}}$.

- The circuit $\mathsf{Circ}^{\mathsf{VerifyEqual}}$ is constructed using $\mathsf{Circ}^{\mathsf{VerifyEqZero}}$, which verifies whether the input wire a_1 equals 0. The circuit $\mathsf{Circ}^{\mathsf{VerifyEqual}}$, on input a_1, a_2 calls $\mathsf{Circ}^{\mathsf{VerifyEqZero}}$ with input $a_1 - a_2$. The circuit $\mathsf{Circ}^{\mathsf{VerifyEqZero}}$ takes as input wire a_1, outputs the indicator wire a_2, which is set to 1 if (and only if) $a_1 = 0$, and 0 otherwise. Furthermore, it makes use of a helper wire a_3 and consists of two sub-circuits:

  1. $\mathsf{Circ}^{\mathsf{AssertEqual}}(a_2 \cdot a_1, 0)$
  2. $\mathsf{Circ}^{\mathsf{AssertEqual}}(a_1 \cdot a_3, 1 - a_2)$

  each of which consists of one constraint (see above). Technically, the circuit $\mathsf{Circ}^{\mathsf{AssertEqual}}$ takes as input just one wire and not a multiplication of two wires. Nevertheless, since it does not use the v values (see above), we can set $v_1 = 1$ for the first sub-circuit and $v_3 = 1$ for the second sub-circuit. On a technical level, the constraints for this circuit are as follows:

  $$(0 \cdot a_0 + 0 \cdot a_1 + 1 \cdot a_2 + 0 \cdot a_3) \cdot (0 \cdot a_0 + 1 \cdot a_1 + 0 \cdot a_2 + 0 \cdot a_3) = 1 \cdot a_0 + 0 \cdot a_1 + 0 \cdot a_2$$
  $$(0 \cdot a_0 + 1 \cdot a_1 + 0 \cdot a_2 + 0 \cdot a_3) \cdot (0 \cdot a_0 + 1 \cdot a_1 + 0 \cdot a_2 + 1 \cdot a_3) = 1 \cdot a_0 + 0 \cdot a_1 + (-1) \cdot a_2$$

  The first constraint ensures that a_1 (the input wire) or a_2 (the indicator wire) is zero. Therefore, in case $a_1 \neq 0$, this constraint forces a_2 to be set to zero. What is left is that if $a_1 = 0$, we must set the indicator wire to 1. For this, the second constraint enforces that $a_1 \cdot a_3 = 1 - a_2$. In case that $a_1 = 0$, this forces to set a_2 to 1. However, there must be a satisfied assignment for the case of $a_1 \neq 0$ for this constraint. We do so via the helper wire a_3: Since the first constraint enforces $a_2 = 0$ in this case, this constraint reads as $a_1 \cdot a_3 = 1$, and thus one can assign a_3 to the inverse of a_1 in $\mathbb{F}$.

- The circuit $\mathsf{Circ}^{\mathsf{VerifyBit}}$ takes computes $a_2 = a_1 \cdot (1 - a_1)$ based on the input wire a_1 and than runs $\mathsf{Circ}^{\mathsf{VerifyEqual}}(a_2, 0)$. Thus, this circuit consists of three constraints, compared to one constraint of $\mathsf{Circ}^{\mathsf{AssertBit}}$.

As these rather basic examples show, there is a massive difference in the number of constraints between verification and assertion. Therefore, our circuits for Kryvos use assertions instead of verifications if possible.

5.3.4.2. Comparisons

A crucial ingredient of elections is the comparison of candidates. Therefore, comparisons play a key role in our circuits, and expressing them through QAP constraints results in a similar picture to our MPC protocols presented in Section 3.2.3. Comparing for equality using $\mathsf{Circ}^{\mathsf{AssertEqual}}$ or $\mathsf{Circ}^{\mathsf{VerifyEqual}}$ is simple, but testing for greater-than ($\geq$) is more complex. We can employ these circuits as sub-circuits for other comparisons, such as $\leq$. For greater-than tests, as in multi-party computation, the bit size of compared values is required, but for equality tests, the circuit size is independent of it.

We will now construct $\mathsf{Circ}^{\mathsf{AssertGt}}$, the circuit that takes as input two wires a_1 and a_2, and asserts that $a_1 \geq a_2$, i.e., the circuit cannot be satisfied if $a_1 < a_2$. The circuit makes use of a split gate [PHGR13] that takes as input a wire a_1 with a value of known bit length n, and additional wires $a_2, \dots a_{2+n}$ with binary values. The split gate asserts that $a_1 = \sum_{i=0}^{n-1} 2^i \cdot a_{2+i}$. This circuit requires $n+1$ constraints: n constraints to show $\forall i \in 0, n-1 : \ a_{2+i} \in \{0,1\}$ and an additional constraint to show the equality of a_1 and $\sum_{i=0}^{n-1} 2^i \cdot a_{2+i}$. This circuit, and the following circuits $\mathsf{Circ}^{\mathsf{AssertGt}}$ and $\mathsf{Circ}^{\mathsf{VerifyGt}}$ make use of the bit length as the setup parameter, rather than an input wire: This requires knowing the bit length of the input values at setup time but allows for more efficient constructions with fewer constraints overall.

The idea of $\mathsf{Circ}^{\mathsf{AssertGt}}$ is as follows: Assuming we are computing in $\mathbb{Z}$, if $a_1 \geq a_2$, then $a_1 - a_2 \geq 0$, and otherwise $a_1 - a_2 < 0$. Translating this idea to $\mathbb{F}$ (where we in the following assume F_p for some prime p) requires restricting the values of the input wires: Assuming that the bit length of a_1 and a_2 is at most $\frac{|\mathbb{F}|}{2}$, we can test whether $a_1 - a_2 \in \{0, \dots, \frac{|\mathbb{F}|}{2} - 1\}$. With that, $\mathsf{Circ}^{\mathsf{AssertGt}}$ consists of a split gate that uses a wire of bit length at most $\frac{|\mathbb{F}|}{2}$ as input and asserts that the value of this wire equals $a_1 - a_2$. The bit size can be selected smaller than $\frac{|\mathbb{F}|}{2}$ to optimize the number of constraints if such conditions apply to a_1 and a_2 during the circuit setup. The circuit $\mathsf{Circ}^{\mathsf{AssertGt}}$ requires $n+2$ constraints, where n denotes the chosen bit length.

The circuit $\mathsf{Circ}^{\mathsf{VerifyGt}}$ verifies that $a_1 \geq a_2$ instead of asserting it, meaning that this circuit outputs a wire a_3 with value 1 if $a_1 \geq a_2$ and value 0 otherwise. This circuit can be satisfied if $a_1 < a_2$, in contrast to $\mathsf{Circ}^{\mathsf{AssertGt}}$. The circuit $\mathsf{Circ}^{\mathsf{VerifyGt}}$ is far more expensive than $\mathsf{Circ}^{\mathsf{AssertGt}}$ regarding the number of necessary constraints. On a high level, the circuit, parameterized by the bit length n, operates as follows: First, it splits a_1 and a_2 into bits. Then, starting with the most significant bit, the circuit iterates over the individual bits. It searches for the first position where the two numbers differ: If the bit of a_1 is 1 at this position, then $a_1 > a_2$, and if the bit of a_1 is 0, then $a_1 < a_2$. If the two numbers do not differ in any position, then $a_1 = a_2$ and therefore $a_1 \geq a_2$. Thus, the circuit searches for the first position at which the two numbers differ, and if the bit of a_1 at this position is 0, then the result is 0. In every other case, the result is 1. In

order to perform this operation in a circuit, several indicator wires are necessary that track the current state of the computation, e.g., storing whether there was a previous position at which the bits differ. Additionally, each iteration step requires an equality verification (comparing the two bits) and several operations updating the indicator wires, which, in terms of circuits, means defining new wires used in the next iteration since values on wires cannot be changed. The circuit needs 11 constraints per bit, resulting in $11 \cdot n$ constraints, where n is the bit length.

5.3.4.3. Circuits for Kryvos

At a high level, our circuits consist of two sub-circuits:

- A cryptographic circuit, denoted by $\mathsf{Circ}^{\mathsf{Com}}$, demonstrating the accuracy of an opening of a commitment.
- A voting circuit, denoted by $\mathsf{Circ}^{f^{\text{res}}}$ or $\mathsf{Circ}^{\mathfrak{C}}$, demonstrating the validity of an election result or ballot.

We refer to the circuits that contain these sub-circuits as $\mathsf{Circ}^{f^{\text{res}}}_{\mathsf{Com}}$ or $\mathsf{Circ}^{\mathfrak{C}}_{\mathsf{Com}}$. The cryptographic component dominates the circuit's size, as we see in Section 5.3.5. For this reason, we will develop an efficient circuit that can verify the accuracy of an opening of a commitment.

A SNARK can prove statements for arbitrary relations. However, the resulting performance (regarding runtime, memory overhead, and bandwidth) quickly deteriorates and becomes impractical for large circuits. For example, Feng et al. [FQZ+21] express neural networks in QAPs. The resulting circuits for small such networks consist of multiple millions of constraints, leading to prover times of multiple minutes and common reference strings (CRSs) of multiple GBs. In contrast, more extensive circuits require over $300,000,000$ constraints, leading to prover times of multiple hours with common reference strings (CRSs) of 100 GB. To perform such experiments, the authors make use of machines with high computational power: They run their evaluations on a Microsoft Azure M32ls instance with 32-core Intel Xeon Platinum 8280M vCPU @ 2.70GHz and 256 GiB DRAM. The work of [KZM+15] proposes the C∅C∅ framework that contains several *SNARK-friendly* cryptographic primitives. For example, they constructed several circuits based on encryption schemes to show the validity of ciphertexts inside a SNARK proof. The resulting circuits to encrypt 200 bytes of plaintext values range between $91,000$ and $868,000$ constraints depending on the encryption scheme.

As the examples above illustrate, including heavy-weight operations (such as cryptography) in circuits requires suitable design and optimization to obtain efficient circuits that we can evaluate using computationally weaker devices, which, in particular, is essential for our setting of e-voting.

Therefore, the main challenge of using SNARKs consists of constructing suitable circuits with minimal numbers of constraints for the intended relations such that the resulting SNARKs are practical. While we discuss the overall circuits for our SNARKs in Section 5.3.5, we need as a central building block for all of our SNARKs an efficient circuit for verifying openings of commitments, which we construct in the following.

5.3.4.4. Commitment Scheme

We use Pedersen commitments over a group $(\mathbb{G}, \cdot)$ of order q with generator g as introduced in [Ped91]. Recall that to compute a Pedersen commitment, one first takes an auxiliary value h defined by $h = g^a$ for a uniform $a \xleftarrow{\$} \mathbb{F}_q$ and then commits to a value $v \in \mathbb{F}_q$ with $\mathsf{Com}(v, r) = g^v h^r$ using a uniform $r \xleftarrow{\$} \mathbb{F}_q \setminus \{0\}$. Pedersen commitments are a standard solution with a simple structure and slight computational complexity. Furthermore, they are additively homomorphic and perfectly hiding.

The choice of the group $\mathbb{G}$ significantly affects the efficiency of the corresponding SNARK, with some groups being more favorable than others. For example, the first natural choice $\mathbb{G} = \mathbb{F}_q$ for a prime q of secure bit length at least $2,048$ is a priori incompatible with the underlying proof system, which supports only values of up to 255 bits. While it is possible to circumvent this problem, e.g., by splitting each value (of at least $2,048$ bits) into chunks of at most 255 bits, the resulting circuit would be too inefficient for our use case: The C∅C∅ framework [KZM+15] realizes RSA encryption with $2,048$ bit keys and requires at minimum $496,000$ constraints to encrypt 200 bytes. Therefore, we use an elliptic curve $\mathbb{G}$, where sufficient security guarantees are available at a much lower length q.

5.3.4.5. SNARKs with Elliptic Curve Circuits

This section presents elliptic curves and discusses how to process computations on an elliptic curve within a Groth16 SNARK.

We will shortly review the definition of an elliptic curve. An elliptic curve E over a finite field $\mathbb{F}_p$ for some prime $p > 3$ is a non-singular algebraic curve of order three in the projective plane $\mathbb{P}_2(\mathbb{F}_p)$. Let E_{aff} be the affine part of E defined by the equation $y^2 = F(x) = x^3 + ax + b$ with $a, b \in \mathbb{F}_p$ such that $4a^3 + 27b^2 \neq 0$, i.e. $E_{\mathsf{aff}} = \{(x, y) \in \mathbb{F}_p^2 \mid y^2 = F(x)\}$. Recall that $E = \{(x_0 : x_1 : x_2) \in \mathbb{P}_2(\mathbb{F}_p) \mid (\frac{x_1}{x_2})^2 = F(\frac{x_0}{x_2})$ and $E_{\mathsf{aff}} = E \setminus O$, $O = (0 : 0 : 1)$.

Given three distinct points A, B, C on E and a line g through these points, one sets $A+B+C = 0$. This additive relation extends uniquely to all of E such that $(E, +)$ becomes an abelian group.

The geometric definition of the addition on E has an algorithmic equivalent, which we will use instead for our implementation:

Theorem 5.5. *a) If $A = (x_A, y_A) \neq (x_B, y_B) = B$ on E_{aff} with $x_A \neq x_B$ then $C = (x_C, y_C) = A + B$ if*

$$x_C = \mu^2 - x_A - x_B$$
$$y_C = -y_A - \mu(x_C - x_A)$$

where $\mu = \frac{y_B - y_A}{x_B - x_A}$. For $x_A = x_B$ and $A \neq B$ one has $A + B = O$.

b) For $A \in E_{\mathsf{aff}}$ with $y_A \neq 0$, $C = 2A \in E_{\mathsf{aff}}$ has coordinates:

$$x_C = \nu^2 - 2x_A$$
$$y_C = -y_A - \nu(x_C - x_A)$$

with $\nu = \frac{F'(x_A)}{2y_A}$. If $y_A = 0$, then $2A = O$.

In order to use the Groth16 SNARK to verify computations on elliptic curves and, in particular, the decryption step, we need to transform the algorithms over elliptic curves into arithmetic circuits over some finite field $\mathbb{F}_p$. To do so, we represent a point $P \in E$ as a triple $(x_P, y_P, z_P) \in \mathbb{F}_p \times \mathbb{F}_p \times \mathbb{F}_2$ with $(x_P, y_P, 0)$ for $P \in E_{\mathsf{aff}}$ and $(0, 0, 1)$ for $P = O$. Hence, we implemented the algorithms for addition (see Theorem 5.5), multiplication (see, e.g., [For14]), as well as more complex algorithms based on these elementary operations on elliptic curves such that they become available to be processed inside the Groth16 SNARK.

We choose specifically Curve25519 which is a Montgomery curve that works over $\mathbb{F}_q$ with $q = 2^{255} - 19$ prime and curve equation $y^2 = x^3 + Ax^2 + x$ for some $A \in \mathbb{F}_q$ with $A^2 \neq 4$. The exact value of A does not affect the efficiency of the SNARK in terms of the number of constraints representing the corresponding arithmetic circuit. Therefore, we can choose it freely (under the requirement $A^2 \neq 4$ of the curve). We use $A = 486,662$ as done in [Ber06]. We can represent the coordinates of this curve using 255 bits and do not need to split values. To stay compatible with the usual notation for Pedersen commitments, we denote the group operation on an elliptic curve with $\cdot$ instead of the usual $+$. The wording *addition* and *multiplication* is consequentially changed to *multiplication* and *exponentiation*.

A Pedersen commitment $g^v \cdot h^r$ requires exponentiations and multiplications, with exponentiations being the most expensive operation. We leverage results from the C∅C∅-framework [KZM+15], which observes that we can implement the highly efficient Montgomery ladder algorithm for exponentiations with relatively few constraints. However, a significant limitation of this approach is that we can not directly multiply the resulting values.

Therefore, we extend this approach to support efficient multiplication of such values. Among others, we achieve this by designing small circuits for the fast y-coordinate recovery algorithm by Okeya and Sakurai [OS01], conversion between projective and affine coordinates, and point multiplication. We construct the overall circuit for computing a Pedersen commitment by efficiently combining these components.

The chosen Montgomery curve allows us to compute the exponentiation of a point g of $\mathbb{G}$ purely based on the first (affine) coordinate of $g = (g_x, g_y) \in \mathbb{F}_q \times \mathbb{F}_q$ and in addition to that we do not require a splitting approach. Furthermore, exponentiation on a Montgomery curve can be realized very efficiently with the ladder algorithm [Mon87, CS18], and this efficiency transfers to the SNARK — an observation already made in [KZM+15]. To fully use the ladder algorithm's efficiency advantage, we must work with projective coordinates for the exponentiation since the resulting circuit does not have to check exceptional cases related to ∞ separately. For other

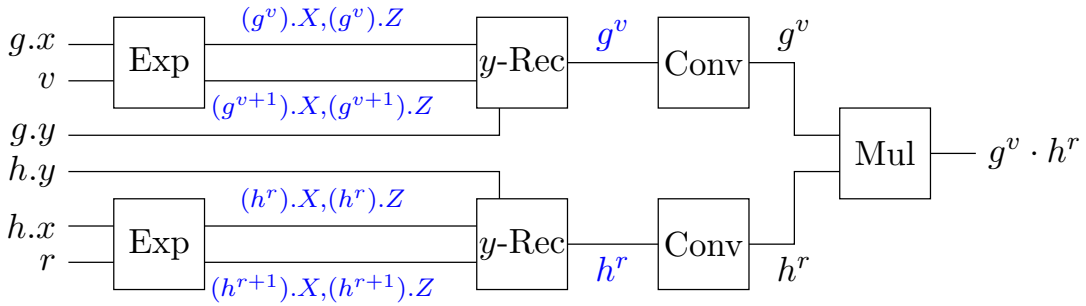


Figure 5.3.: Circuit of a Pedersen commitment. By $g.x$ and $g.y$ we denote the affine x-coordinate and the affine y-coordinate of g, while upper case letters denote the respective projective coordinates. Additionally, projective coordinates are illustrated in blue.

operations, however, e.g., for multiplication, we use affine coordinates. Generally, we chose for each algorithm the representation that allows the most efficient execution of this specific algorithm.

In order to obtain an efficient and secure instantiation of the Pedersen commitment scheme, we will use a subgroup of an elliptic curve group for $(\mathbb{G}, \cdot)$. Note that for computing a commitment inside the SNARK, we need to be able to express the exponentiations g^v and h^r and the computation of their product with as few constraints as possible.

Unlike previous results, e.g., the C∅C∅-framework, which focuses on a Diffie-Hellman key exchange inside a SNARK, Pedersen commitments do not only require exponentiations of points inside the SNARK but also the multiplication of different curve points g^v and h^r in order to compute $\mathsf{Com}(v, r) = g^v \cdot h^r$. For this multiplication, however, we need the y-coordinate of both g^v and h^r, which are not provided by the Montgomery ladder. However, there is a standard way to reconstruct a y-coordinate by Okeya and Sakurai [OS01]. Their algorithm (y-Rec) takes the output of a Montgomery ladder, which is apart from $(g^v)_x$ also $(g^{v+1})_x$, as well as both coordinates of the original input g, and outputs the y-coordinate of $(g^v)_y$.

After we recover both coordinates, we perform the before-mentioned switch from projective to affine coordinates to prepare for the final multiplication of g^v and h^r. Then, we compute the multiplication of g^v and h^r in the standard way for Montgomery curves.

Altogether, we get the circuit illustrated in Figure 5.3 for the computation of a Pedersen commitment.

Table 5.1 shows micro-benchmarks of these operations inside the SNARK. Essentially, the recovery of the y-coordinate, the conversion, and the point multiplication are negligible compared to the complexity of the Montgomery ladder. These benchmarks also show that a Montgomery curve is a suitable choice in our setup since the advantage of the efficient exponentiation on this curve easily compensates for the additional complexity of the Okeya and Sakurai algorithm.

In principle, we have now obtained a way to represent a Pedersen commitment through an arithmetic circuit. However, as Table 5.1 shows, even with our design choices, representing

Operation	Constraints
Montgomery Ladder [Mon87] (255 bits Exponent)	5,084
Montgomery Ladder [Mon87] (32 bits Exponent)	624
y-Coordinate Recovery (Okeya-Sakurai Algorithm [OS01])	39
Point Multiplication	86
Conversion (Projective to Affine Coordinates)	15
Pedersen Commitment (v of 255 bits, r of 255 bits)	10,360
Pedersen Commitment (v of 32 bits, r of 255 bits)	5,900

Table 5.1.: Number of (QAP) constraints for various operations.

a single Pedersen commitment still costs 11,701 constraints, which still turns out to be too expensive for our application.

Hence, we will now discuss further optimizations.

5.3.4.6. Further Performance Improvements

Even with the introduced design choices, the resulting arithmetic circuit for computing a commitment is still too large for our purposes. Hence, we will now introduce further necessary optimizations that drastically reduce the number of constraints required for computing the commitments that correspond to $c_{\mathsf{agg}}^{\mathsf{choice}}$ (recall that in the basic approach, one commitment contains the number of votes for one choice, so we represent $c_{\mathsf{agg}}^{\mathsf{choice}}$ by $n_{\mathsf{components}}$ commitments). From Table 5.1, we can see that even when using a Montgomery curve and the Montgomery ladder, the main factor for the complexity of computing a Pedersen commitment inside a SNARK is still the exponentiation of g^v and h^r. However, recall that in our application, the input v is the aggregated number of votes for a particular candidate/choice. Hence, v will typically be relatively small and not require the entire 255 bits (for example, in a single-vote election, v will not exceed the number of eligible voters). As indicated by Table 5.1, limiting v to 32 bits already almost halves the constraints required to express the computation of a Pedersen commitment inside the SNARK. Note that the circuit also ensures that the input values are of the correct size. Regarding the randomness r, however, we will need an exponentiation with up to 255 bits for security reasons.

Next, recall that as part of our system, we want to compute commitments corresponding to the tally $c_{\mathsf{agg}}^{\mathsf{choice}}$, where $c_{\mathsf{agg}}^{\mathsf{choice}}$ is a list of the aggregated votes $c_{i,\mathsf{agg}}^{\mathsf{component}}$ for each candidate/choice $c_i^{\mathsf{component}}$. If one straightforwardly implements this via standard Pedersen commitments, then there will be one separate commitment $\mathsf{com}(c_{i,\mathsf{agg}}^{\mathsf{component}}, r_i)$ per $c_{i,\mathsf{agg}}^{\mathsf{component}}$ (just as Helios uses one ciphertext per choice). Each of these commitments introduces another exponentiation for some randomness r_i to obtain h^{r_i}, which must also be computed in the arithmetic circuit to prove knowledge of $c_{\mathsf{agg}}^{\mathsf{choice}}$. Since each randomness requires the full 255 bits, this can become very

costly in elections with multiple choices/candidates (see the left side of Table 5.2). Instead, we solve this issue using Pedersen vector commitments described in [BG18]. Essentially, these commitments allow for committing to a vector $\boldsymbol{v} = (v_1, \ldots, v_N)$ with entries in $\mathbb{F}_q$ by computing

$$\mathsf{Com}(\boldsymbol{v}, r) = g_1^{v_1} \cdot \ldots \cdot g_N^{v_N} \cdot h^r$$

for generators $g_1, \ldots, g_N$ of $\mathbb{G}$ chosen uniformly at random. In this case, N is the *slot size* of the commitment. Pedersen vector commitments are also additively homomorphic, perfectly hiding, and computationally binding under the discrete logarithm assumption [ACC+22]. By setting N to the number of choices, this construction requires just one exponentiation with randomness of 255 bits to commit to the full set of choices. Using vector commitments drastically improves the computational cost and the size of EKCRS and VKCRS as shown in Table 5.2. For committing to 250 values, moving from computing one commitment per value to computing one commitment for all values in a single SNARK reduces the number of required constraints by more than 1.2 million, which drastically reduces the proof time as well as the size of EKCRS. Regarding the size of VKCRS, we note that the difference between the two circuits is as follows: Using N many Pedersen commitments requires the circuit to consist of N many Pedersen commitments as public wires, which leads to increasing sizes of VKCRS with increasing slot size. In contrast, using a single Pedersen vector commitment for N many values results in a single commitment as a public wire. Thus, the size of VKCRS is independent of the slot size due to the construction of the Pedersen vector commitment, which always consists of a single group element, regardless of the number of plaintext values.

Overall, the above optimizations efficiently prove knowledge of an opening utilizing SNARKs. We employ this technique to prove knowledge of $c_{\mathsf{agg}}^{\mathsf{choice}}$ and to allow voters to prove the validity of their ballot. We use both applications in Kryvos, as shown in Section 5.3.5.

We have implemented these optimizations, resulting in a library with optimized representations via constraints of many fundamental elliptic curve operations, available at [HKK+23]. Hence, this library should be helpful beyond our work, e.g., to obtain an efficient QAP instance of ElGamal.

5.3.4.7. The Optimal Number of Slots per Commitment

The slot size of the Pedersen vector commitments used in Kryvos mainly influences the circuit size of the SNARKs that prove the commitments' correct opening. These SNARKs appear in two parts of Kryvos:

1. The SNARKs for proving ballot validity in the voting phase. Each voter creates such SNARK proofs.

2. The SNARK in the evaluation phase is used to prove the correct computation of the result function on the aggregated tally. The designated trustee creates this proof. The SNARK

Choices/N	N Pedersen Commitments				1 Pedersen Vector Commitment			
	Constraints	Prove [s]	EKCRS [MB]	VKCRS [kB]	Constraints	Prove [s]	EKCRS [MB]	VKCRS [kB]
1	5,990	0.148	3.871	2.666	5,990	0.148	3.871	2.666
5	29,950	0.741	19.355	4.574	9,246	0.229	5.975	2.666
10	59,900	1.482	38.710	6.959	13,316	0.329	8.605	2.666
25	149,750	3.704	96.776	14.114	25,526	0.631	16.496	2.666
50	299,500	7.409	193.552	26.039	45,876	1.135	29.647	2.666
100	599,000	14.817	387.104	49.889	86,576	2.142	55.950	2.666
150	898,500	22.226	580.656	73.739	127,276	3.148	82.252	2.666
200	1,198,000	29.634	774.207	97.589	167,976	4.155	108.554	2.666
250	1,497,500	37.043	967.759	121.439	208,676	5.162	134.857	2.666

Table 5.2.: Comparison of Pedersen commitments and Pedersen vector commitments using a single Groth16 SNARK. This table uses inputs v of 32 bits.

for the evaluation always takes the complete aggregated tally as input and computes the election result based on the values of the aggregated tally.

One natural question is determining the slot size N used by the vector commitments to optimize the computation of these SNARKs. Table 5.3 shows the number of constraints per circuit and Figure 5.4 the benchmarks of monolithic Groth16 SNARKs (i.e., we use one single SNARK proof to prove the correctness of the openings of $n_{\mathsf{components}} = 100$ values) in dependency of the slot size N per commitment (which is equal to $100/\#commitments$ in this case). For $n_{\mathsf{tuples}} < n_{\mathsf{components}}$, such a monolithic SNARK needs to be computed by the trustees to prove the correct computation of the result function on the aggregated tally. The voters, however, can create an individual SNARK proof for each commitment. Note that using commitments of different slot sizes would require the voters to use a different EKCRS for each commitment of a different size. For this reason, if we want to use multiple commitments and multiple SNARKs, we always use commitments of equal slot size and adapt the relation to be proved in a way such that the voters need exactly one CRS for all SNARK proofs (see also Section 5.3.5 for more details on this relation).

Each additional commitment adds complexity to the computation of the SNARK, resulting in costly operations for large slot sizes. While in Table 5.2 we only considered the extreme cases of $N = 1$ and $N = n_{\mathsf{components}}$, we see that the benchmarks in Figure 5.4 confirm the intuition that for proving the correct opening using a monolithic SNARK it is, in general, more efficient to use fewer commitments of larger slot size than using many commitments of smaller slot size for the same number of values. Thus, for the tallying SNARK (which is always a monolithic SNARK), we want to use the maximum slot size $N = n_{\mathsf{components}}$ such that all choices fit into a single commitment.

Number of Slots per Commitment	Constraints
1	654,900
2	371,550
4	229,875
5	201,540
10	144,870
20	116,535
25	110,868
50	99,534
100	93,867

Table 5.3.: Number of constraints of circuits that show the opening commitments of various slot sizes to 100 values.

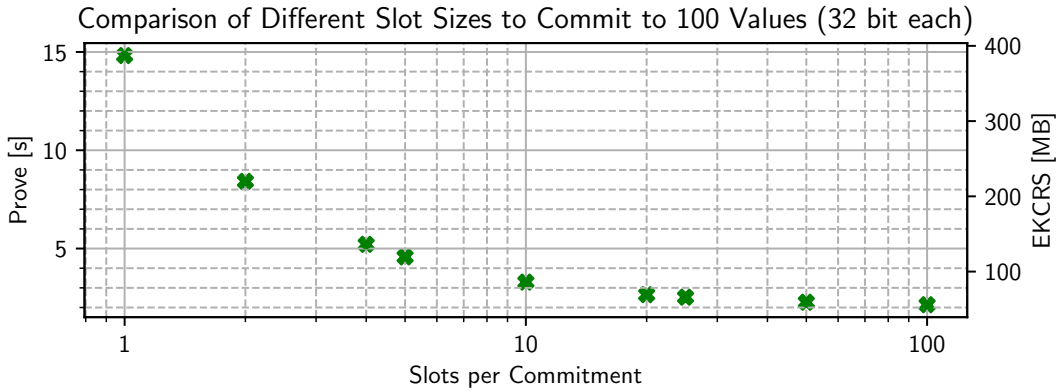


Figure 5.4.: Benchmarks of SNARKs that compute commitment(s) with different slot sizes to 100 values. Each point represents a single SNARK that uses commitments with the given number of slots to commit to 100 values.

A crucial property of our voting system is that voters can efficiently create ballots. Thus, we require that creating the SNARK proof of the ballot's validity is practical for the voters. As the benchmarks in Section 5.3.5 will show, using one Pedersen vector commitment of maximal slot size containing the ballot inside a monolithic SNARK proof is usually also efficient enough for proving the ballot's validity. However, one exception to this is Instant-Runoff Voting (IRV), where $n_{\mathsf{components}}$ can become exceptionally high, and hence the size of EKCRS as well as the time necessary to create a proof using a monolithic SNARK can become impractical. In these cases, we opt for letting voters use multiple commitments of smaller slot size and multiple SNARK proofs even though these multiple commitments only allow for less efficient tallying SNARKs. However, as we will describe in Section 5.3.5, e.g., using a slot size of $N = n_{\mathsf{components}}/10$ allows

for computing the SNARK proofs of ballot validity as well as the monolithic tallying SNARK efficiently.

5.3.5. Case Study

We provide a proof of concept implementation of Kryvos that we have instantiated for a wide range of simple and complex voting methods and various result functions; our implementation is available at [HKK+23]. We use the libsnark library [sci17] for the Groth16 SNARK. While in Section 5.3.1 we gave a high-level description of Kryvos, there are various ways in how Kryvos can be implemented, involving, among others, fixing a slot size N, a suitable choice space, designing and optimizing circuits for both ballot and tallying SNARKs.

We have instantiated and evaluated Kryvos for various voting methods and result functions. As it turns out, there are essentially two main requirements for a voting method to be supported by Kryvos in practice:

1. The voting method must support a ballot format allowing homomorphic ballot aggregation. Aggregation is necessary to protect the privacy of voters against the trustees.

2. It must be possible to define a choice space to efficiently create the resulting ballots, including the corresponding ballot SNARKs.

We note that a voting method defines a choice space required to prove ballot validity. More precisely, proving ballot validity includes proving that a vote belongs to the respective choice space $\mathfrak{C}$. However, an election result function f^{res} does not (only) depend on the voting method and its corresponding choice space. Some result functions only make sense for specific choice spaces. For systems like single-vote and multi-vote, we consider various result functions that, e.g., compute only the candidate with the most votes or all candidates that gained at least a certain threshold of votes. The chosen result function only influences the tallying SNARK but not the ballot SNARK.

We obtained the following benchmarks using an ESPRIMO Q957 (64bit, i5-7500T CPU @ 2.70GHz, 16 GB RAM). We do not use parallelism; we use just a single core to make the results independent of the specific core count, making it easier to compare with other benchmarks. Of course, in practice, one would parallelize, e.g., if multiple SNARK proofs need to be computed, thereby reducing the overall runtime depending on the number of cores available.

The main efficiency concerns of the setting of e-voting are regarding ballot submission: voters must be able to cast their ballots efficiently, even when using devices with low computational power and memory. In contrast, the trustees typically have access to servers with far more resources to evaluate the tally. Thus, the primary goal of our Kryvos instantiations is to allow for efficient ballot casting. We set the limit using the device described above. A voter can cast a ballot in under one minute, using a EKCRS of the size of at most one GB.

5.3.5.1. Choice Spaces

In principle, we can instantiate the Kryvos framework with other existing ZKPs for ballot correctness designed for a specific choice space, where such ZKPs are available (e.g., [CDS94]). To show the validity of a ballot (for all voting methods except IRV), a voter computes a single Groth16 SNARK that takes the single Pedersen vector commitment of slot size $N = n_{\mathsf{components}}$ obtained by aggregating the commitments on all vote shares of a voter as public input, the opening as secret input, and then proves that the opening is from the correct choice space and corresponds to the commitment.

We now define the concrete relations for the SNARKs showing ballot validity, depending on the respective choice space: Let $c_i^{\mathsf{choice}} = (c_{1,i}^{\mathsf{component}}, \ldots, c_{n_{\mathsf{components}},i}^{\mathsf{component}})$ be the choice of the voter. As specified by Auth, depending on the voting function, this choice is split into n_{tuples} blocks, each of size N. Thus, the actual committed value is $(t^{i,1}, \ldots, t^{i,n_{\mathsf{tuples}}})$. Let $(c^{i,1}, \ldots, c^{i,n_{\mathsf{tuples}}})$ be these commitments with randomness vector $(r^{i,1}, \ldots, r^{i,n_{\mathsf{tuples}}})$.

The following relation describes that a vector $(c^{i,1}, \ldots, c^{i,n_{\mathsf{tuples}}})$ is a (vector) commitment to a valid choice $c_i^{\mathsf{choice}} \in \mathfrak{C}$ with randomness vector $(r^{i,1}, \ldots, r^{i,n_{\mathsf{tuples}}})$:

$$\begin{aligned}\mathcal{R}_{\mathfrak{C}} = \{((c^{i,1}, \ldots, c^{i,n_{\mathsf{tuples}}}), ((t^{i,1}, \ldots, t^{i,n_{\mathsf{tuples}}}), (r^{i,1}, \ldots, r^{i,n_{\mathsf{tuples}}}))) \mid \\ \forall j \in \{1, \ldots, n_{\mathsf{tuples}}\} : c^{i,j} = \mathsf{Com}(t^{i,j}; r^{i,j}) \wedge (t^{i,1}, \ldots, t^{i,n_{\mathsf{tuples}}}) \in \mathfrak{C}\}.\end{aligned}$$

We denote the circuit for $\mathcal{R}_{\mathfrak{C}}$ with $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}}$. This circuit consists of two sub-circuits: $\mathsf{Circ}^{\mathsf{Com}}$ verifies the commitment, while $\mathsf{Circ}^{\mathfrak{C}}$ verifies that the plaintext is a valid choice. Furthermore, we denote the SNARK for this relation by $\Pi_{\mathfrak{C}}$. This relation shows that the number of public wires for this relation is independent of $\mathfrak{C}$ but only depends on n_{tuples}, as the only public wires are regarding the commitments. Using Pedersen vector commitments, each of the n_{tuples} commitments consists of a single elliptic curve group element and thus requires three public values. The following relations shown by SNARKs will use a single commitment as input. Thus, using Groth16 SNARK, every ballot validity SNARKs can be verified in $\sim$2.8 ms using a VKCRS of size about 2.666 kB.

In the following, we will present concrete instantiations with the choice spaces that Kryvos currently supports.

5.3.5.2. Single Choice

The most spartan and widely used choice space that Kryvos supports is $\mathfrak{C}_{\mathsf{Single}}$. Here, the choices symbolize the election's candidates. For this voting method, each voter has exactly one vote she can give to her preferred candidate. The circuit to show validity of the choice $(c_{j,i}^{\mathsf{component}})_{j=1}^{n_{\mathsf{components}}}$ of voter v_i, that is, showing $(c_{j,i}^{\mathsf{component}})_{j=1}^{n_{\mathsf{components}}} \in \mathfrak{C}_{\mathsf{Single}}$ is done in two parts:

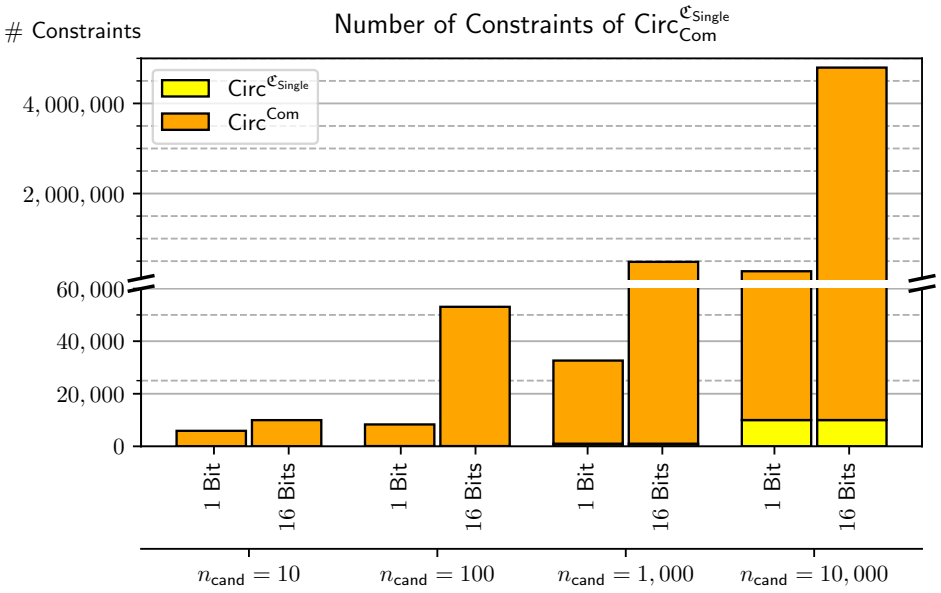


Figure 5.5.: Number of constraints of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}_{\mathsf{Com}}$.

1. For each $c^{\mathsf{component}}_{j,i}$ we assert $c^{\mathsf{component}}_{j,i} \in \{0,1\}$. This assertion is done via the circuit $\mathsf{Circ}^{\mathsf{AssertBit}}$ for each (choice) component of the ballot, leading to 1 constraint per (choice) component.

2. The circuit checks that $\sum_{j=1}^{n_{\mathsf{components}}} c^{\mathsf{component}}_{j,i} = 1$. We do so by running $\mathsf{Circ}^{\mathsf{AssertEqual}}$ as a sub-circuit which asserts that $\sum_{j=1}^{n_{\mathsf{components}}} c^{\mathsf{component}}_{j,i}$ equals 1. Recall that $\mathsf{Circ}^{\mathsf{AssertEqual}}$ requires 1 constraint.

In summary, with $n_{\mathsf{components}}$, the circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}$ requires $n_{\mathsf{components}} + 1$ constraints. In Figure 5.5, we show the number of constraints of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}_{\mathsf{Com}}$. We can choose a bit size of a single bit for the wires in the case of single-choice. Since the circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}$ asserts that all wires are binary, we only need to perform one step of the Montgomery ladder algorithm for the commitment. Therefore, for comparison, we present two versions of the circuit, with 1-bit and 16-bit values. The bit size determines the number of bits the circuit has to extract and directly determines the number of steps the circuit needs to execute. The bit size of the values determines the number of constraints of the commitment. However, both $\mathsf{Circ}^{\mathsf{AssertBit}}$ and $\mathsf{Circ}^{\mathsf{AssertEqual}}$ do not depend on the size of the inputs' bits. Therefore, bit size does not affect the circuit used to check $\mathfrak{C}_{\mathsf{Single}}$. The figure also shows how much of the overall constraints of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}_{\mathsf{Com}}$ are part of the sub-circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}$. As the figure indicates, $\mathsf{Circ}^{\mathsf{Com}}$ is the primary factor of the constraints of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}_{\mathsf{Com}}$.

Furthermore, in Figure 5.6, we present benchmarks of the respective Groth16 SNARKs of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{Single}}}_{\mathsf{Com}}$ for single bit values, which is sufficient for this choice space. As the benchmarks indicate, proving the validity of a ballot is highly efficient, even for immense numbers of candidates. For 30,000 candidates, the size of EKCRS is about 500 MB.

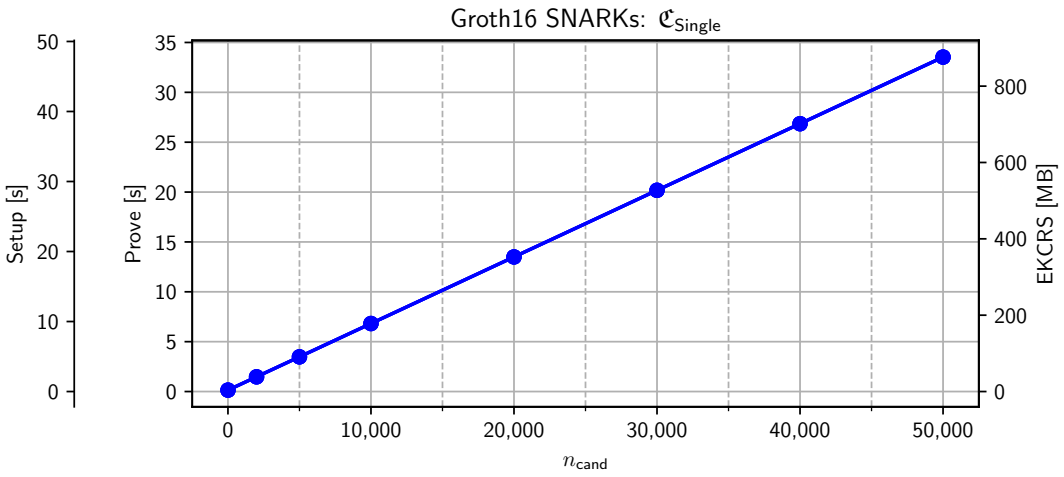


Figure 5.6.: Benchmarks of Groth16 SNARKs of $\Pi_{\mathfrak{C}_{\mathsf{Single}}}$.

5.3.5.3. Multiple Choice

The choice space $\mathfrak{C}_{\mathsf{Multi},\mathfrak{o}_{\mathsf{comp}},n_{\mathsf{votes}},b}$ of multi-vote consists of multiple parameters:

- The parameter b denotes how many votes the voter can assign to a specific candidate.
- The parameter n_{votes} denotes how many votes the voter can distribute overall among the candidates.
- The operator $\mathfrak{o}_{\mathsf{comp}}$ denotes whether the voter needs to distribute n_{votes} votes precisely or if this value denotes an upper bound of the voters the voter can distribute.

In order to cover various options and values for these parameters, we construct a generic circuit that uses additional input wires to set the exact options of the specific multiple-choice choice space. The circuit takes three public wires as additional input, each corresponding to one of the abovementioned parameters. Since these wires are public, everyone can check whether voters set these options correctly. The circuit uses these wires as *flags* that determine how the circuit verifies.

The circuit proceeds as follows:

- For each $c_{j,i}^{\mathsf{component}}$ we assert using $\mathsf{Circ}^{\mathsf{AssertGt}}$ that $c_{j,i}^{\mathsf{component}} \leq b$.
- We verify using $\mathsf{Circ}^{\mathsf{VerifyEqual}}$ whether $\sum_{j=1}^{n_{\mathsf{components}}} c_{j,i}^{\mathsf{component}} = n_{\mathsf{votes}}$ and store the result in one wire. In another wire, we store the result of the check $\sum_{j=1}^{n_{\mathsf{components}}} c_{j,i}^{\mathsf{component}} \leq n_{\mathsf{votes}}$.
- The two wires from the previous step are used to compute the overall verification wire based on $\mathfrak{o}_{\mathsf{comp}}$.

Figure 5.7 presents the number of constraints of $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Multi},\mathfrak{o}_{\mathsf{comp}},n_{\mathsf{votes}},b}}$. As the numbers of constraints indicate, the circuit showing $c_i^{\mathsf{choice}} \in \mathfrak{C}_{\mathsf{Multi},\mathfrak{o}_{\mathsf{comp}},n_{\mathsf{votes}},b}$ requires more constraints than the circuit for $\mathfrak{C}_{\mathsf{Single}}$, but its contribution to the overall circuit is again negligible.

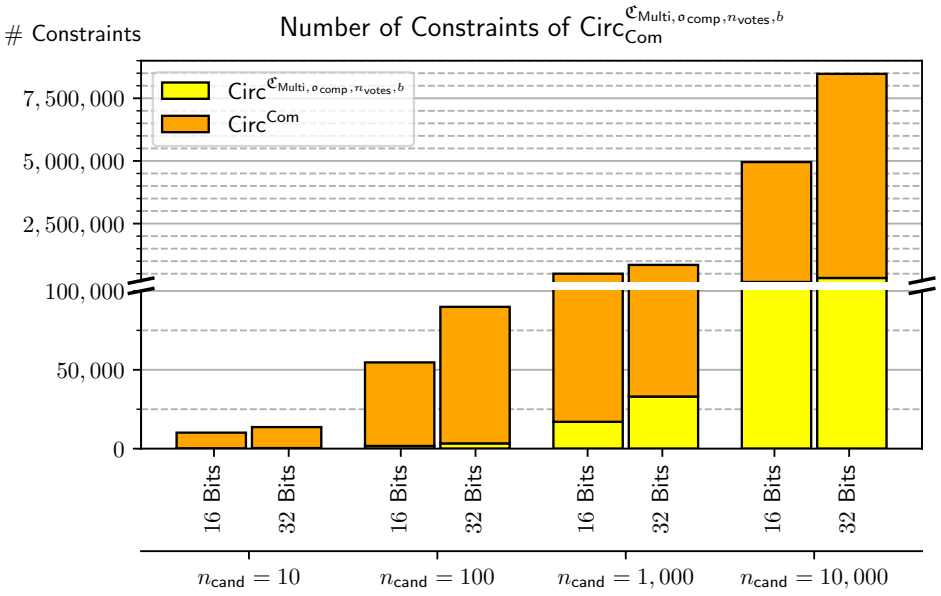


Figure 5.7.: Number of constraints of $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Multi},o_{\mathsf{comp}},n_{\mathsf{votes}},b}}$.

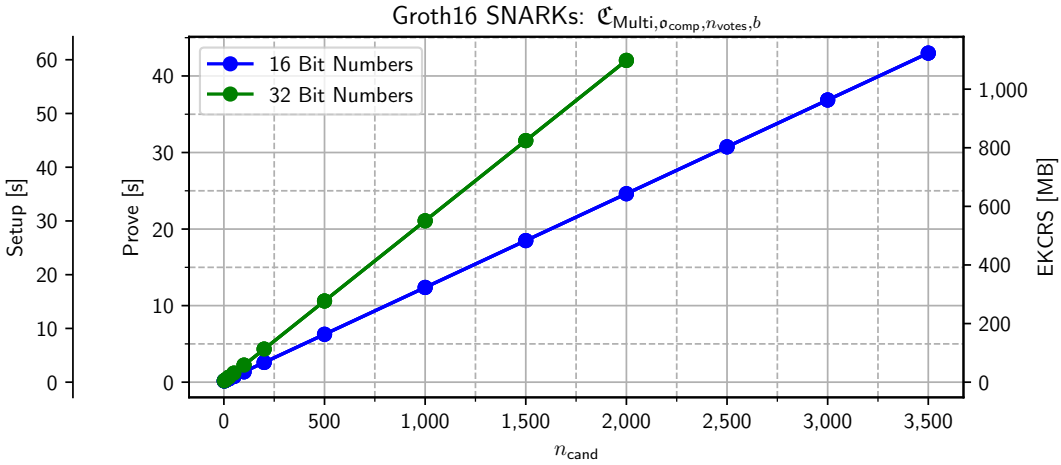


Figure 5.8.: Benchmarks of Groth16 SNARKs of $\Pi_{\mathfrak{C}_{\mathsf{Multi},o_{\mathsf{comp}},n_{\mathsf{votes}},b}}$.

We present the benchmarks of Groth16 SNARKs evaluating these circuits in Figure 5.8. As these benchmarks show, voters can prove ballot validity with $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Multi},o_{\mathsf{comp}},n_{\mathsf{votes}},b}}$ efficiently for more than $1,000$ candidates.

5.3.5.4. Borda

For Borda voting, we present circuits for $\mathfrak{C}_{\mathsf{BordaPointList}}$ and $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$. To our knowledge, we are the first to propose NIZKPs for this choice space.

The circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{BordaPointList}}}$ works as follows. It iterates through all possible points, starting with the highest number of points and going downwards. For each number of points, it checks

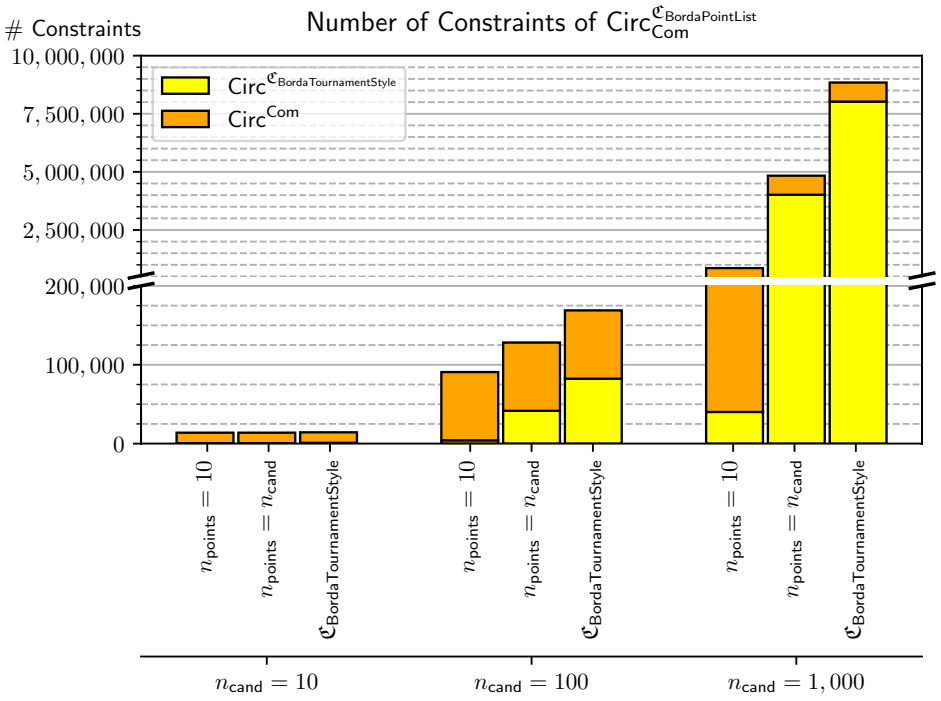


Figure 5.9.: Number of constraints of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{BordaPointList}}}_{\mathsf{Com}}$.

how many candidates this number of points received. If there are n candidates with this number of points, the voter can not distribute the following $n-1$ points. The circuit validates the ballot if this checking routine produces no errors.

The circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{BordaTournamentStyle}}}$ is a particular case of the circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{BordaPointList}}}$ using $\{0, \ldots, 2 \cdot (n_{\mathsf{cand}} - 1)\}$ as point list. Given an input vector $v = (v_1, \ldots, v_{n_{\mathsf{cand}}})$, the circuit iterates through $1, \ldots, n_{\mathsf{cand}}$, and for index i computes the correct number of points assigned to candidate i as the number of entries in v being equal to v_i plus 2 times the number of entries in v smaller than v_i. The resulting circuit grows quadratically in n_{cand} and remains reasonably small for large sizes of candidates. For example, for $n_{\mathsf{cand}} = 100$ candidates, the circuit corresponds to less than $100,000$ constraints.

Besides the number of candidates, the size of this circuit depends on the size of the point list n_{points} since each number of points requires one step in the iteration. Therefore, for our benchmarks, we constructed three different circuits. The first circuit uses a point list of constant size, namely ten different points. The second point list uses $n_{\mathsf{points}} = n_{\mathsf{cand}}$, and the third applies $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$.

We expect that the circuit with $n_{\mathsf{points}} = n_{\mathsf{cand}}$ scales somewhat quadratically in the number of candidates: There are n_{cand} many iteration steps, and each step requires a search for the current number of points among n_{cand} many values. In comparison, the number of constraints of the circuit with $n_{\mathsf{points}} = 10$ should be much lower.

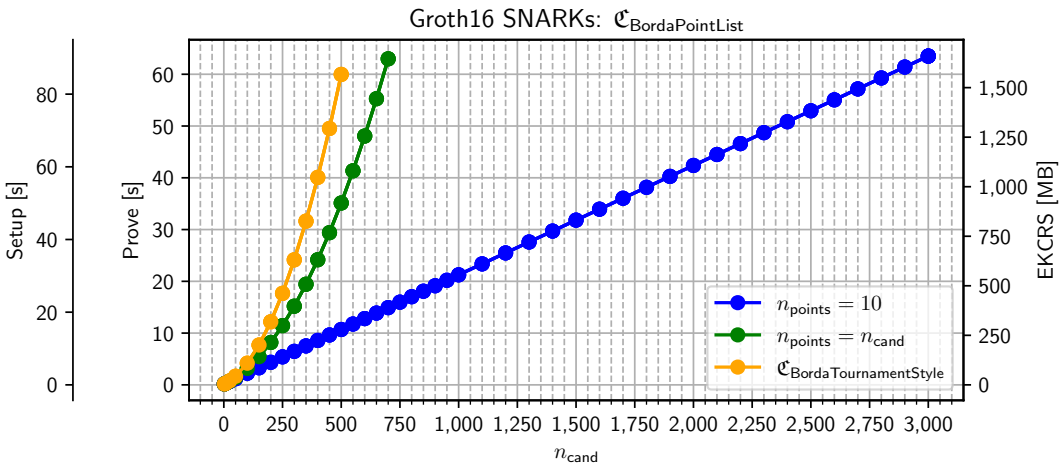


Figure 5.10.: Benchmarks of Groth16 SNARKs of $\Pi_{\mathfrak{C}_{\mathsf{BordaPointList}}}$.

In Figure 5.9 shows the number of constraints of these circuits for various numbers of candidates. This figure underlines our expected behavior: While the number of constraints of the circuit with $n_{\mathsf{points}} = 10$ is relatively small, using $n_{\mathsf{points}} = n_{\mathsf{cand}}$ demands far more constraints. Even in contrast to the number of constraints of the commitments, the constraints of checking $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$ take most of the combined circuit.

The quadratic scaling is also visible in Figure 5.10, where we present the benchmarks of the Groth16 SNARK evaluating these circuits.

In summary, Kryvos can efficiently prove ballot validity regarding $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$ for 500 candidates, and with way more if we use a point list of size independent of the number of candidates.

5.3.5.5. Grading

A choice of the choice space $\mathfrak{C}_{\mathsf{MajorityJudgment}}$ consists of a $n_{\mathsf{cand}} \times n_{\mathsf{grades}}$ matrix where each column consists of a single choice that selects a grade for the corresponding candidate. Therefore, as Figure 5.11 shows, the numbers of constraints are close to the ones of $\mathfrak{C}_{\mathsf{Single}}$. These benchmarks always use values of a single bit. We present two variations: the first uses $n_{\mathsf{grades}} = n_{\mathsf{cand}}$: this leads to a quadratic scaling of the number of constraints of the circuit in the number of candidates. The second variant uses $n_{\mathsf{grades}} =$. The benchmarks in Figure 5.12 include a third variation: using $n_{\mathsf{grades}} = \frac{n_{\mathsf{cand}}}{2}$.

5.3.5.6. Ranking Matrix

The circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{RankingMatrix}}}$ directly applies the definition of this choice space: First, the circuit asserts that all values are binary. Then, for every pair of candidates c_i^{cand} and c_j^{cand} it is asserted that $\psi_{ij} + \psi_{ji} = 1$. Furthermore, for every additional candidate c_k^{cand}, it is asserted that $\psi_{ij} \wedge \psi_{jk} \wedge (1 - \psi_{ik})$ evaluated to false, where the value 0 is interpreted as false, and the value

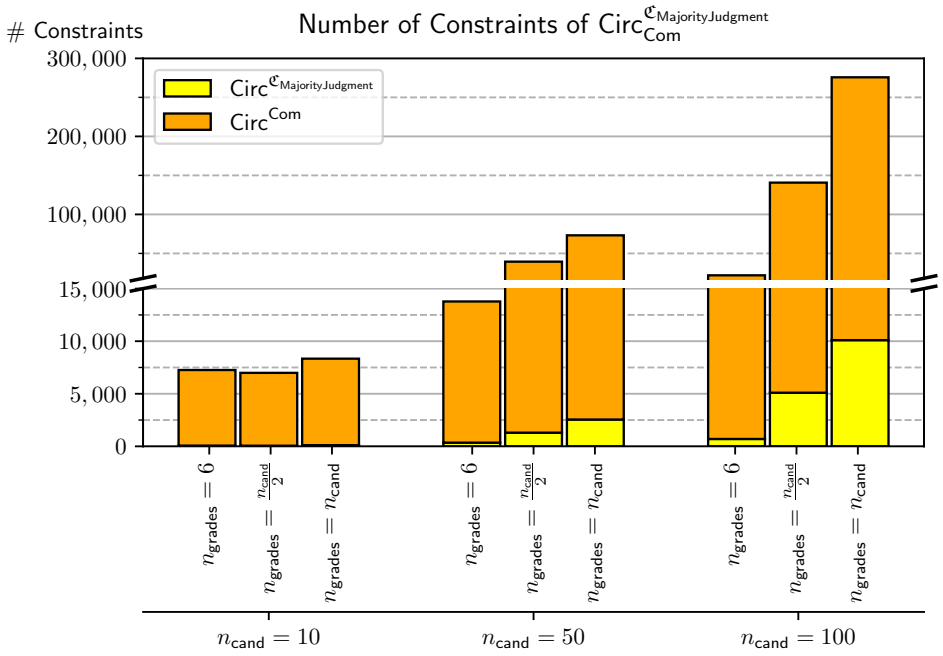


Figure 5.11.: Number of constraints of $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{MajorityJudgment}}}$.

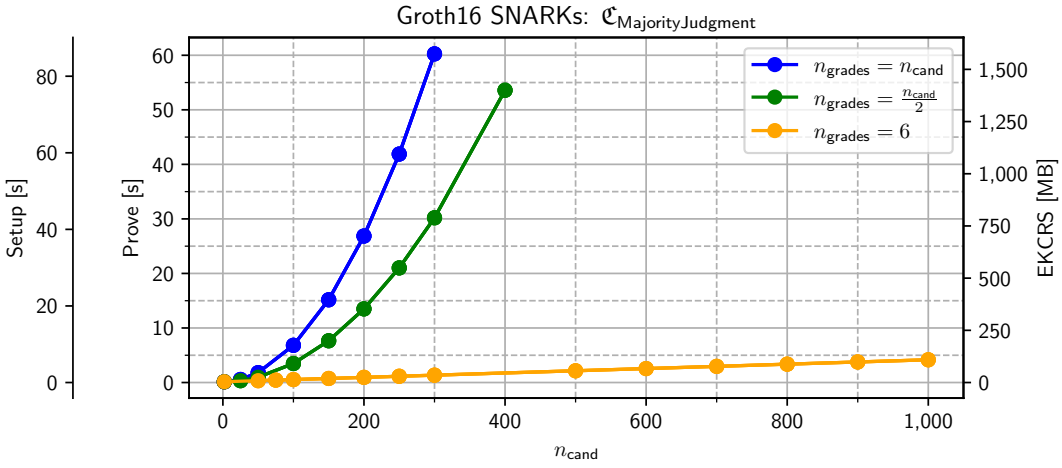


Figure 5.12.: Benchmarks of Groth16 SNARKs of $\Pi_{\mathfrak{C}_{\mathsf{MajorityJudgment}}}$.

1 as true. For this, we use a circuit $\mathsf{Circ}^{\mathsf{AssertAnd}}$ that computes the $\wedge$ value of wires of binary value. For n input wires, $\mathsf{Circ}^{\mathsf{AssertAnd}}$ asserts with one constraint that $\sum_{i=1}^{n} a_i = n$. Overall, as Figure 5.13 shows, the circuit for $\mathfrak{C}_{\mathsf{RankingMatrix}}$ scales cubic in the number of candidates and is not influenced by the bit length of the values on the wires. However, as the number of values to commit to also scales cubic in the number of candidates, the circuit solely related to the voting part, namely checking $\mathsf{b}_i \in \mathfrak{C}_{\mathsf{RankingMatrix}}$ is of negligible size, in comparison to the circuit related to the commitments. However, as the circuit $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{RankingMatrix}}}$ already checks that the input values are binary, we only have to perform one step of the Montgomery ladder algorithm for the

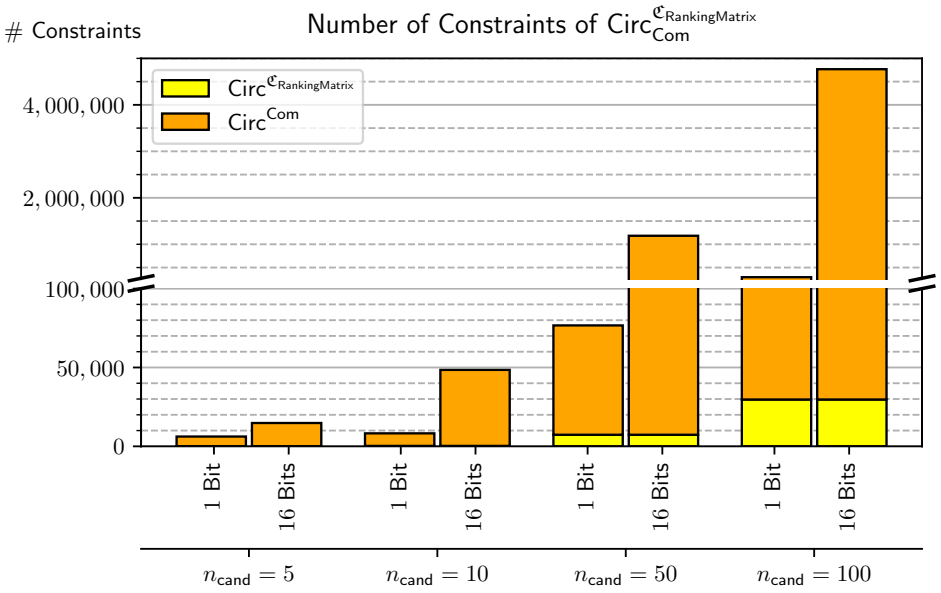


Figure 5.13.: Benchmarks of Groth16 SNARKs of $\mathsf{Circ}^{\mathfrak{C}_{\mathsf{RankingMatrix}}}_{\mathsf{Com}}$.

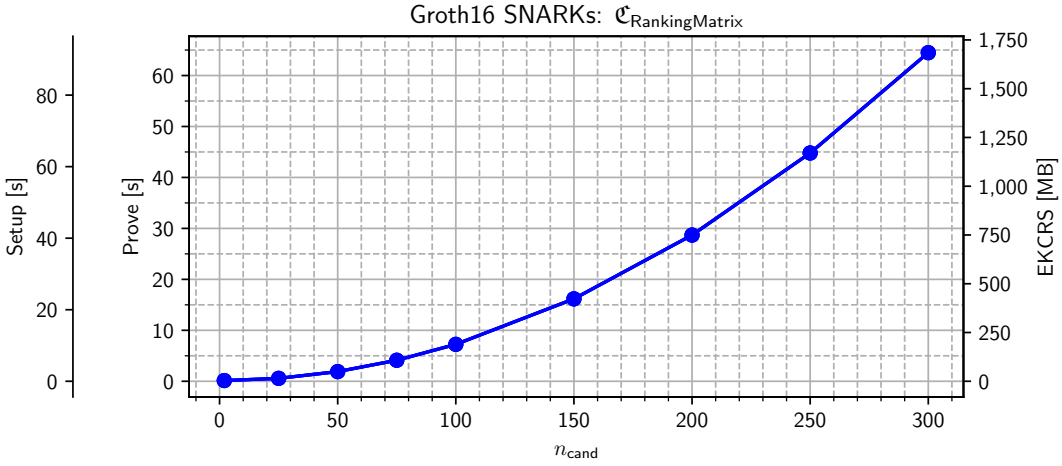


Figure 5.14.: Benchmarks of Groth16 SNARKs of $\Pi_{\mathfrak{C}_{\mathsf{RankingMatrix}}}$.

commitment. Figure 5.14 shows that Kryvos can efficiently handle more than 100 candidates for $\mathfrak{C}_{\mathsf{RankingMatrix}}$. Compared to $\mathfrak{C}_{\mathsf{Single}}$ or b, the number of candidates is smaller. However, elections with this choice space usually have way fewer candidates. For example, the Debian Project (see Section 2.8) runs this choice space with up to 7 candidates, which Kryvos efficiently supports.

5.3.5.7. Ranking Permutation

To realize $\mathfrak{C}_{\mathsf{RankingPermutation}}$ with Kryvos, we make use of the SNARKs for the choice space $\mathfrak{C}_{\mathsf{Single}}$ (see Section 2.6). In the following, we differentiate between complete and partial rankings, as presented in Section 2.6, and we will always use values of 32 bit in this section.

The choice space $\mathfrak{C}_{\mathsf{RankingPermutation}}$ grows exponentially in the number of candidates and thus quickly leads to situations where voters cannot prove the validity of their ballots in a reasonable time (for the slot size $N = n_{\mathsf{components}}$). For example, in the case of a complete ranking of $n_{\mathsf{cand}} = 6$ and hence $n_{\mathsf{components}} = 720$, it is still manageable for voters (the ballot SNARK requires under 15 seconds to compute using a EKCRS of size about 350 MB). In comparison, using a partial ranking for $n_{\mathsf{cand}} = 6$ and hence $n_{\mathsf{components}} = 1,957$ computing the ballot SNARK already requires a EKCRS of size over one GB, which is not considered acceptable by our requirements. One option is to try and improve this situation by replacing the ballot SNARKs with traditional specialized ZKPs optimized for the choice space $\mathfrak{C}_{\mathsf{Single}}$. However, as we show in Section 5.4, this approach only improves ballot ZKP runtime for small-scale slot sizes N close to 1, in which case the runtime of the tallying SNARK becomes entirely impractical.

However, we can improve the runtime of the ballot SNARK by splitting the ballot into a small number of parts and then running multiple SNARKs in parallel. Specifically, encoding a ballot via $n_{\mathsf{tuples}} > 1$ separate commitments, each having slot size $N \approx n_{\mathsf{components}}/n_{\mathsf{tuples}}$, one can show the validity of a ballot belonging to $\mathfrak{C}_{\mathsf{Single}}$ via multiple sub-statements, where each statement is concerned only with one commitment, namely:

(a) For each commitment $c^{i,j}, j \in \{1, \ldots, n_{\mathsf{tuples}} - 1\}$: All slots $x_{(j-1)\cdot N+1}, \ldots, x_{j\cdot N}$ in $c^{i,j}$ are either 0 or 1.

(b) For $c^{i,n_{\mathsf{tuples}}}$: the first slots $x_{(n_{\mathsf{tuples}}-1)\cdot N+1}, \ldots, x_{n_{\mathsf{components}}}$ are either 0 or 1, and all remaining slots $x_{n_{\mathsf{components}}+1}, \ldots, x_{n_{\mathsf{tuples}}\cdot N}$ are 0.

(c) For $t^{i,\mathsf{agg}} := \sum_{j=1}^{n_{\mathsf{tuples}}} c^{i,j}$: The commitment contains a single slot with value 1 and all other slots are 0.

While (a), (b), and (c), in conjunction, prove the validity of a ballot in ZK, they do not prove knowledge of a witness. Intuitively, this is because one can re-arrange the components of a ballot with valid proofs to create a new ballot with valid proofs, even without knowing the contents of the commitments. However, Kryvos only requires a zero-knowledge proof of correctness in order to be secure (see Section 5.3.2). Alternatively, one could add the position of each commitment as additional public input.

If implemented naively, each of the statements (a), (b), and (c) from above requires its own Groth16 SNARK instance and hence its own EKCRS. Note, however, that all of these Groth16 SNARK instances are for quite similar functions: they first check that the secret input w is a valid opening for the public commitment com by re-computing that commitment and then also show that w has specific properties. As shown in Figure 5.5 for $\mathfrak{C}_{\mathsf{Single}}$, re-computing the commitment accounts for almost the total size of the EKCRS, whereas performing any additional checks on w barely affect the size of the EKCRS. Hence, we can use the following optimization: we consider a Groth16 SNARK for a function that takes as public input not just the commitment com but also an additional flag $\mathit{flag} \in \mathbb{N}$. This Groth16 SNARK instance allows for creating a

# Commitments ($= n_{\mathsf{tuples}}$)	1	2	3	5	10
Slot Size ($= N$)	1,957	979	653	392	196
Constraints	1,600,138	803,068	537,378	324,663	164,923
EKCRS [MB]	1,034.09	518.98	347.28	209.81	106.58
VKCRS [kB]	3.14	3.14	3.14	3.14	3.14
Prove [s]	39.58	19.87	13.29	8.03	4.08
Verify [ms]	5.51	5.51	5.51	5.51	5.51
Prove Overall [s]	39.58	59.60	53.17	48.19	44.88

Table 5.4.: Comparison of various slot sizes for $\mathfrak{C}_{\mathsf{RankingPermutation}}$ with $n_{\mathsf{cand}} = 6$ and thus $n_{\mathsf{components}} = 1,957$.

# Commitments ($= n_{\mathsf{tuples}}$)	1	2	10	20	50
Slot Size ($= N$)	5,040	2,520	504	252	101
Constraints	4,112,783	2,058,983	415,943	210,563	87,498
EKCRS [MB]	2,657.89	1,330.62	268.80	136.08	56.55
VKCRS [kB]	3.14	3.14	3.14	3.14	3.14
Prove [s]	101.74	50.93	10.29	5.21	2.16
Verify [ms]	5.51	5.51	5.51	5.51	5.51
Prove Overall [s]	101.74	152.80	113.18	109.38	110.38

Table 5.5.: Comparison of various slot sizes for $\mathfrak{C}_{\mathsf{RankingPermutation}}$ with $n_{\mathsf{cand}} = 7$ (complete ranking) and thus $n_{\mathsf{components}} = 5,040$.

single proof that shows that, on the one hand, the secret input w is an opening for com, and on the other hand, depending on *flag*, that w fulfills various additional properties. If $n_{\mathsf{components}}$ is not a multiple of N, we could also let the last commitment have a smaller slot size than the previous ones. However, each voter would need an additional EKCRS to create a proof for (b). Thus, if $n_{\mathsf{components}}$ is not a multiple of N, we artificially increase the slot size of the last commitment in order to be able to use a single CRS for the ballot SNARK.

Using this method, we must determine a suitable slot size N (and hence corresponding n_{tuples}). A smaller N allows for higher parallelism up to the number of CPU cores available, improving runtime for the ballot SNARK. At the same time, a smaller N increases the combined runtime of all ballot sub-SNARKs (as shown in Section 5.3.4 but also since (c) introduces an additional commitment opening over a new aggregated commitment, which is not necessary if a single SNARK shows all statements) and of the single tallying SNARK. Note that the tallying SNARK is monolithic and thus cannot use parallelism.

# Commitments ($= n_{\mathsf{tuples}}$)	5	10	15	20	50
Slot Size ($= N$)	2,740	1,370	914	685	274
Constraints	2,238,283	1,121,733	750,093	563,458	228,493
EKCRS [MB]	1,446.49	724.92	484.75	364.13	147.66
VKCRS [kB]	3.14	3.14	3.14	3.14	3.14
Prove [s]	55.37	27.75	18.55	13.94	5.65
Verify [ms]	5.51	5.51	5.51	5.51	5.51
Prove Overall [s]	332.20	305.22	296.87	292.70	288.26

Table 5.6.: Comparison of various slot sizes for $\mathfrak{C}_{\mathsf{RankingPermutation}}$ with $n_{\mathsf{cand}} = 7$ and thus $n_{\mathsf{components}} = 13,700$.

Since, as shown above, we can handle 720 choices for $\mathfrak{C}_{\mathsf{Single}}$ with $n_{\mathsf{tuples}} = 1$ and thus support 6 candidates for complete rankings and 5 candidates for partial rankings (leading to 326 choices for $\mathfrak{C}_{\mathsf{Single}}$), we want to explore which additional numbers of candidates we can support using $n_{\mathsf{tuples}} > 1$. In Table 5.4 we present benchmarks for $n_{\mathsf{cand}} = 6$ with a partial ranking, in Table 5.5 benchmarks for $n_{\mathsf{cand}} = 7$ with a complete ranking, and in Table 5.6 benchmarks for $n_{\mathsf{cand}} = 7$ with a partial ranking. These tables show the benchmarks for various possible slot sizes for $\mathfrak{C}_{\mathsf{RankingPermutation}}$. The row *SNARK:* Prove indicates the runtime for computing a single substatement and hence also indicates the overall runtime if all statements (1 statement for $n_{\mathsf{tuples}} = 1$ and $n_{\mathsf{tuples}} + 1$ statements otherwise due to the addition of (c)) can be computed in parallel. The row *SNARK:* Prove *Overall* gives the combined sequential runtime.

As Table 5.4 shows, even if we use only $n_{\mathsf{tuples}} = 2$ and hence $N \approx n_{\mathsf{components}}/2$, then a voter can construct a ballot in less than a minute using an EKCRS of size of under 520 MB, which is manageable. The performance progressively improves with increasing n_{tuples}. The voter can use parallelism to compute multiple SNARKs simultaneously to decrease the time it takes to create proof showing ballot validity.

Tables 5.5 and 5.6 show that using the technique of increasing n_{tuples} allows us to drastically reduce the size of the EKCRS. However, we cannot get a sequential time of generating all SNARK proofs under one minute. A voter can create a ballot using three cores and show its validity under one minute for $n_{\mathsf{cand}} = 7$ and a complete ranking if she computes all three SNARKs in parallel. The voter cannot apply this optimization for $n_{\mathsf{cand}} = 7$ and a partial ranking since the voter needs way more cores and thus cannot efficiently create ballots on devices with low computational power. Furthermore, it is impossible to efficiently prove ballot validity for $n_{\mathsf{cand}} = 8$ in the case of a complete ranking since this requires even more slots than $n_{\mathsf{cand}} = 7$ for a partial ranking, namely 40,320.

Choice Space	Parameters	n_{cand}	
		1 s	30 s
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Single}}}$	N/A	1,288	44,709
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Multi},o_{\mathsf{comp}},n_{\mathsf{votes}},b}}$	16 Bit Numbers	71	2,439
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Multi},o_{\mathsf{comp}},n_{\mathsf{votes}},b}}$	32 Bit Numbers	41	1,425
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{BordaPointList}}}$	$n_{\mathsf{points}} = 10$	41	1,413
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{BordaPointList}}}$	$n_{\mathsf{points}} = n_{\mathsf{cand}}$	36	455
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{BordaTournamentStyle}}}$	N/A	32	339
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{MajorityJudgment}}}$	$n_{\mathsf{grades}} = n_{\mathsf{cand}}$	35	215
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{MajorityJudgment}}}$	$n_{\mathsf{grades}} = \frac{n_{\mathsf{cand}}}{2}$	51	291
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{MajorityJudgment}}}$	$n_{\mathsf{grades}} = 6$	7	6,519
$\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{RankingMatrix}}}$	N/A	35	200

Table 5.7.: Maximum number of candidates for various choice spaces that Kryvos can evaluate in under one and 30 seconds. We present the results for $\mathfrak{C}_{\mathsf{RankingPermutation}}$ in Tables 5.4 to 5.6.

5.3.5.8. Summary

In this section, we present the findings of the ballot ZKPs of Kryvos and how the GPPS enables instantiations for various choice spaces. We offer instantiations not only for choice spaces where specialized ZKP solutions already exist (e.g., $\mathfrak{C}_{\mathsf{Single}}$ and $\mathfrak{C}_{\mathsf{Multi},o_{\mathsf{comp}},n_{\mathsf{votes}},b}$) but also for choice spaces where no such alternatives currently exist (e.g., $\mathfrak{C}_{\mathsf{BordaTournamentStyle}}$) or where the alternative specialized ZKPs have downsides. For instance, the specialized ZKP for $\mathfrak{C}_{\mathsf{RankingMatrix}}$ requires further computation by the trustees beyond the voter's participation, which our Kryvos instantiation does not need.

Our efficient Kryvos instantiations provide reasonable ballot ZKP creation times for voters, even for many candidates. We summarize the benchmarks in Table 5.7, which show the number of candidates a voter can handle in an election, given a time limit on the ballot ZKP creation time. For a Groth16 SNARK proof with a creation time of one second on our machine, the EKCRS consists of about 25 MB and about 785 MB for a proof creation time of EKCRS seconds.

The table shows that even with proof creation times of under a single second, voters can cast ballots for reasonable numbers of candidates.

5.3.5.9. Tallying for Various Result Functions

The tallying phase mainly consists of aggregating the ballots and computing the final Groth16 SNARK proof showing the election result's correctness. Aggregation of Pedersen (vector) commitments is high-speed, and the trustees can already start the aggregation in the previous voting phase. Hence, our benchmarks' driving factor and focus lies in the final creation of Groth16 SNARK. For all result functions, the tallying SNARK essentially takes the aggregated commitment(s) on the tally and the election result as public input, the commitment opening(s) as secret input, and then proves that the opening corresponds to the commitment and that the election result was obtained by applying f^{res} to the tally contained in the opening. We present our instantiations and benchmarks for various election result functions in the following.

5.3.5.10. Election Result Functions

This section presents our instantiations of the Kryvos framework for various election result functions, including their benchmarks. Before we dive into specific instantiations of the Kryvos framework, we first define the generic relation regarding the election result function.

Let $x = (c^{\mathsf{agg},1}, \ldots, c^{\mathsf{agg},n_{\mathsf{tuples}}}, \mathsf{elecres})$ and $w = (c^{\mathsf{choice}}_{\mathsf{agg}}, r^{\mathsf{agg},1}, \ldots, r^{\mathsf{agg},n_{\mathsf{tuples}}})$. By $t^{\mathsf{agg},i}$ we denote the n_{tuples} tuples that represent the tally for the commitments: $(t^{\mathsf{agg},1}, \ldots, t^{\mathsf{agg},n_{\mathsf{tuples}}}) = c^{\mathsf{choice}}_{\mathsf{agg}}$.

The following relation describes that the final result elecres is correctly computed w.r.t. the commitments $(c^{\mathsf{agg},1}, \ldots, c^{\mathsf{agg},n_{\mathsf{tuples}}})$ and result function f^{res}.

$$\begin{aligned}\mathcal{R}^{\text{res}}_f = \{((c^{\mathsf{agg},1}, \ldots, c^{\mathsf{agg},n_{\mathsf{tuples}}}, \mathsf{elecres}), (c^{\mathsf{choice}}_{\mathsf{agg}}, r^{\mathsf{agg},1}, \ldots, r^{\mathsf{agg},n_{\mathsf{tuples}}})) \mid \\ (t^{\mathsf{agg},1}, \ldots, t^{\mathsf{agg},n_{\mathsf{tuples}}}) = c^{\mathsf{choice}}_{\mathsf{agg}}, \mathsf{elecres} = f^{\text{res}}(c^{\mathsf{choice}}_{\mathsf{agg}}), \\ \forall i \in \{1, \ldots, n_{\mathsf{tuples}}\} \colon c^{\mathsf{agg},i} = \mathsf{Com}(t^{\mathsf{agg},i}; r^{\mathsf{agg},i})\}\end{aligned}$$

We denote the SNARK for this relation by $\Pi_{f^{\text{res}}}$. As in the case of $\mathcal{R}_{\mathfrak{C}}$, we remark that the number of public wires of $\mathcal{R}^{\text{res}}_f$ scales in n_{tuples}. Furthermore, in contrast to $\mathcal{R}_{\mathfrak{C}}$, the relation $\mathcal{R}^{\text{res}}_f$ also includes the election result elecres as public input. The number of public wires also depends on elecres. Typically, the election result consists of a single wire per candidate, indicating whether the candidate won. We recall that in general, $n_{\mathsf{cand}} \leq n_{\mathsf{components}}$. Therefore, the size of VKCRS and the time of Verify scales in the number of candidates.

5.3.5.11. Plurality

The election result function $f^{\text{res}}_{\mathsf{Plurality}}$ outputs the candidates that received the most votes. Kryvos realizes this election result function with a circuit that finds the maximum value $m_{\max}$ in a list l and outputs the indices of this maximum value. For this, the prover first computes the maximum value $m_{\max}$ of l, and then the circuit asserts, using $\mathsf{Circ}^{\mathsf{AssertGt}}$, that $m_{\max}$ is greater

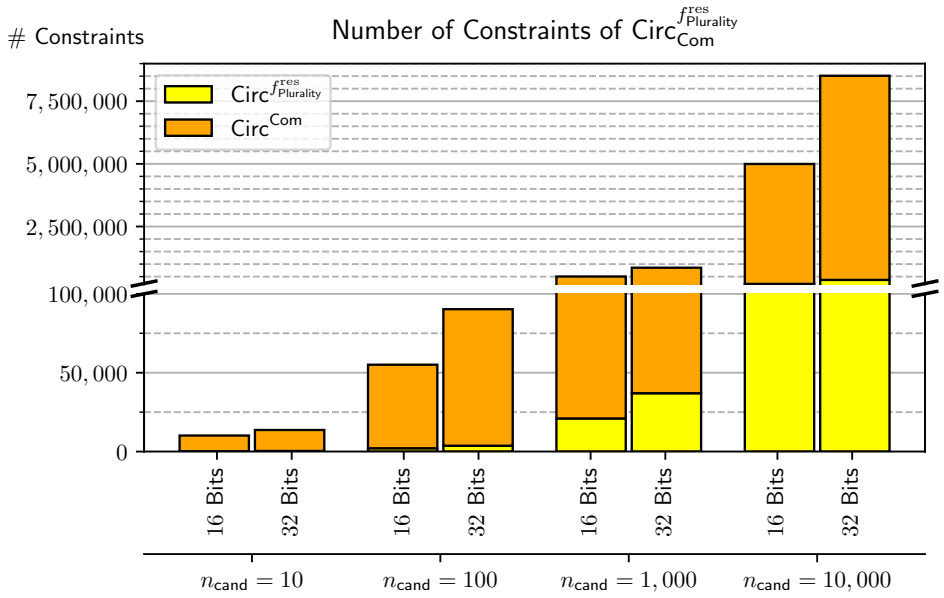


Figure 5.15.: Number of constraints of $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{Plurality}}}_{\mathsf{Com}}$.

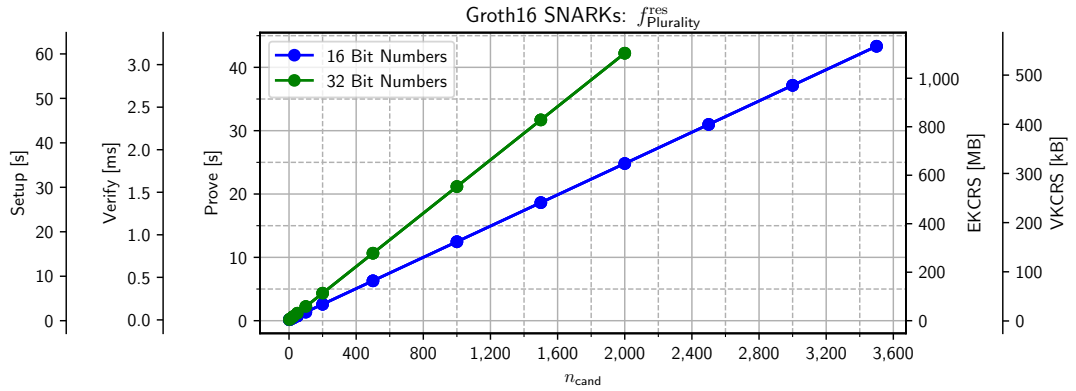


Figure 5.16.: Benchmarks of Groth16 SNARKs of $\Pi_{f^{\mathrm{res}}_{\mathrm{Plurality}}}$.

than every value in l. Next, for each item l_i in the list, an indicator wire is set according to $\mathsf{Circ}^{\mathsf{VerifyEqual}}(l_i, m_{\max})$. These indicator wires form the public output of the circuit and the election result.

In Figure 5.15, we present the number of constraints of $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{Plurality}}}_{\mathsf{Com}}$. These numbers show that the circuit mainly consists of constraints of $\mathsf{Circ}^{\mathsf{Com}}$, and only a small percentage is of $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{Plurality}}}$.

In Figure 5.16, we present the benchmarks of the Groth16 SNARKs evaluating $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{Plurality}}}_{\mathsf{Com}}$. As the figure shows, the benchmarks scale linearly in the number of candidates, and Kryvos can easily handle $2,000$ candidates efficiently.

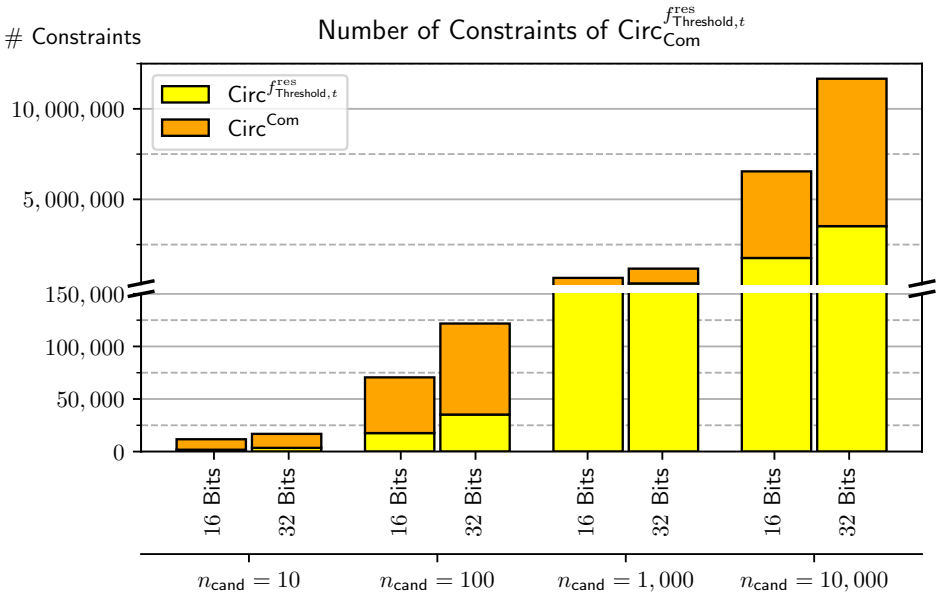


Figure 5.17.: Number of constraints of $\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$.

5.3.5.12. Threshold

The circuit $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$ checks for every candidate whether this candidate received at least t many votes. On a conceptual level, this procedure is similar to the procedure used to check for $\mathfrak{C}_{\mathsf{Multi},\mathfrak{o}_{\mathsf{comp}},n_{\mathsf{votes}},b}$ that the vote for every candidate is at most b. However, on a technical level, these two circuits need to be constructed differently: In the case of $\mathfrak{C}_{\mathsf{Multi},\mathfrak{o}_{\mathsf{comp}},n_{\mathsf{votes}},b}$, the circuit asserts that every value is at most b since the voter cannot cast a ballot with invalid entries. In contrast, for $f^{\mathsf{res}}_{\mathsf{Threshold},t}$, candidates might not receive sufficient votes. Therefore, the circuit for this relation cannot assert operations for this part: The circuit would not be satisfiable if a candidate did not receive sufficient votes.

This difference between the circuits for choice spaces and election result functions is present in many cases where the concepts of circuits for specific choice spaces appear similar to those of circuits for specific election result functions. While the circuits for choice spaces can use assertions, the circuits for election result functions must use comparisons instead, leading to a higher number of constraints in comparison.

In direct comparison, as presented in Section 5.3.4, the circuit $\mathsf{Circ}^{\mathsf{AssertGt}}$ requires less constraints than $\mathsf{Circ}^{\mathsf{VerifyGt}}$. However, the number of constraints of the circuit for $f^{\mathsf{res}}_{\mathsf{Threshold},t}$ is independent of the value t. In Figure 5.17, we present the number of constraints of this circuit. This figure shows that the computation of the election result nearly takes up to a third of the overall constraints of the circuit.

In Figure 5.18, we present the benchmarks of the Groth16 SNARKs $\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$. As the figure shows, the benchmarks scale linearly in the number of candidates, and Kryvos can easily handle $2,000$ candidates efficiently.

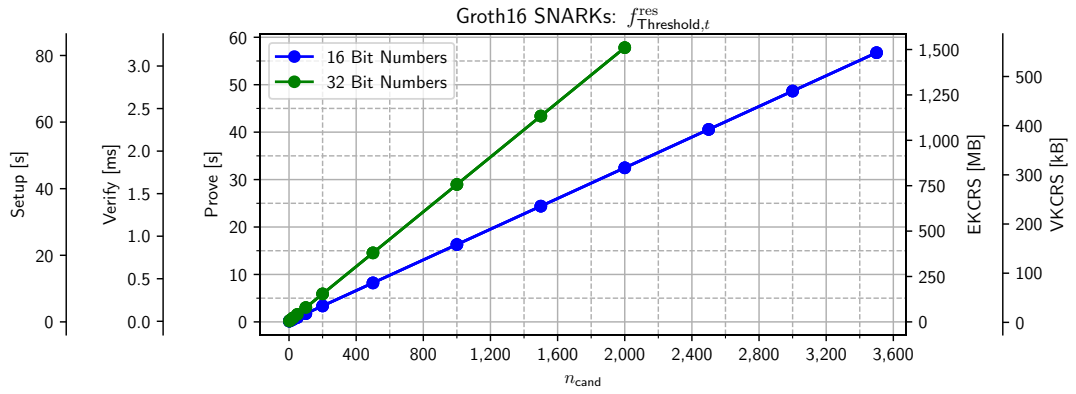


Figure 5.18.: Benchmarks of Groth16 SNARKs of $\Pi_{f^{\text{res}}_{\text{Threshold},t}}$.

5.3.5.13. Best

The circuit $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}$ that computes $f^{\text{res}}_{\text{Best},n}$ works as follows. First, the prover computes the number of votes n_{votes} such that each winning candidate received at least n_{votes} votes. The circuit uses this value as additional secret input and proceeds as follows. Using $\mathsf{Circ}^{f^{\text{res}}_{\text{Threshold},t}}$ as a sub-circuit, $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}$ asserts that the set of candidates that received at least n_{votes} votes is of size at least n. In order to prove that this set is indeed the set of winning candidates, the circuit additionally uses $\mathsf{Circ}^{f^{\text{res}}_{\text{Threshold},t}}$ again as a sub-circuit to assert that the set of candidates that received at least $n_{\text{votes}} + 1$ votes is smaller than n. So, at its core, the circuit $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}$ consists of two sub-circuits $\mathsf{Circ}^{f^{\text{res}}_{\text{Threshold},t}}$. Therefore, apart from some constraints usable by both sub-circuits (e.g., the split gates for the votes), the complexity of $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}$ is about two times the complexity of $\mathsf{Circ}^{f^{\text{res}}_{\text{Threshold},t}}$

Figure 5.19 shows the constraints of $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}_{\mathsf{Com}}$. The figure shows that the sub-circuit $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}$ underlines the complexity to $\mathsf{Circ}^{f^{\text{res}}_{\text{Threshold},t}}$. In comparison to $\mathsf{Circ}^{\mathsf{Com}}$, the number of constraints of $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}$ increases faster in the number of candidates, to a point where the circuit $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}_{\mathsf{Com}}$ takes about half of the constraints for 32 bit values and $n_{\mathsf{cand}} = 10,000$.

We present the benchmarks of Groth16 SNARKs evaluating $\mathsf{Circ}^{f^{\text{res}}_{\text{Best},n}}_{\mathsf{Com}}$ in Figure 5.20.

5.3.5.14. Majority Judgment Median

The circuit $\mathsf{Circ}^{f^{\text{res}}_{\text{MJMedian}}}$ computes for each candidate the median grade. We can compute the median value in a list as follows. The list consists of values $v_1, \ldots, v_{n_{\text{grades}}}$, where v_i denotes how many votes grade i received. The median i_{med} is the index of the first grade in the list such that $\sum_{i=1}^{i_{\text{med}}} v_i \geq \frac{n_{\text{votes}}}{2}$, i.e., at least half of the overall votes n_{votes} are in $v_1, \ldots, v_{i_{\text{med}}}$.

The circuit verifies the median grade as follows. First, the prover computes the index i_{med} of the median grade in plain. Then, the circuit asserts that $\sum_{i=1}^{i_{\text{med}}} v_i \geq \frac{n_{\text{votes}}}{2}$. In order to check that this is indeed the correct index of the median, the circuit additionally asserts that $\sum_{i=1}^{i_{\text{med}}-1} v_i$ is smaller than $\frac{n_{\text{votes}}}{2}$.

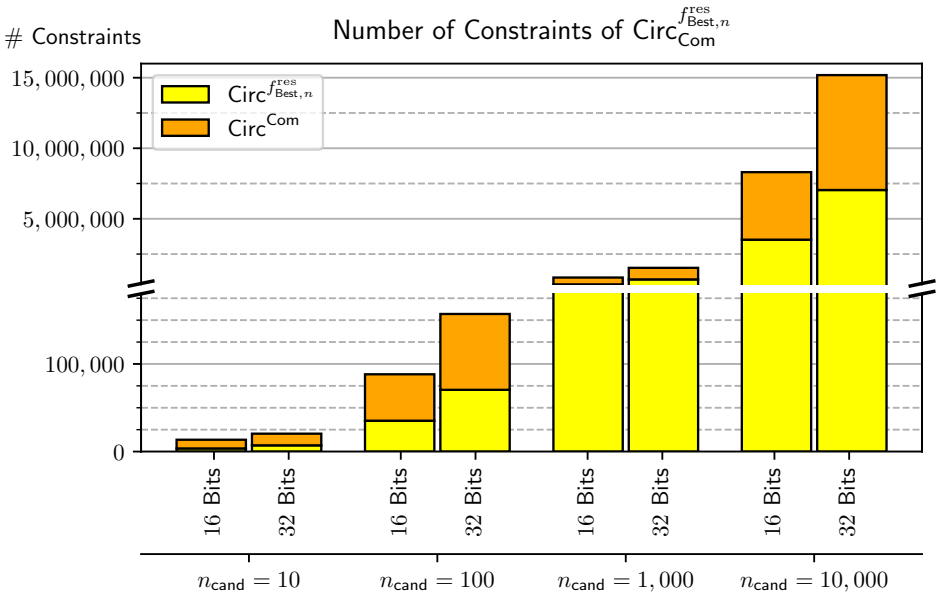


Figure 5.19.: Number of constraints of $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{Best},n}}_{\mathsf{Com}}$.

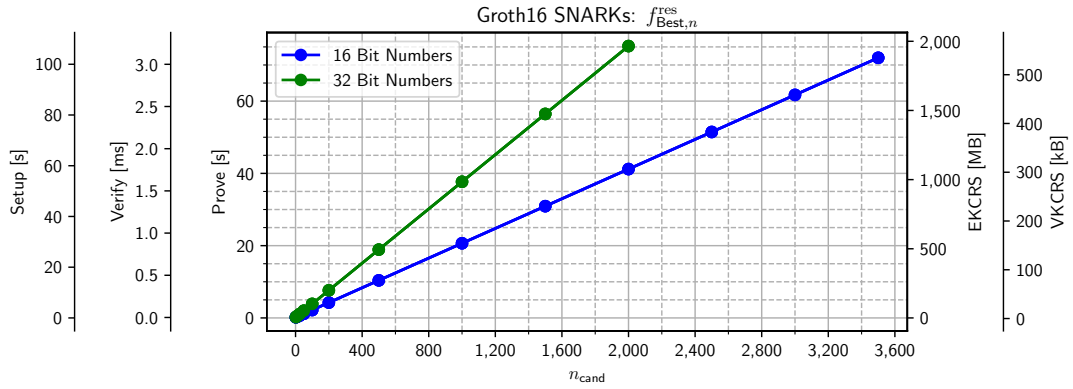


Figure 5.20.: Benchmarks of Groth16 SNARKs of $\Pi_{f^{\mathrm{res}}_{\mathrm{Best},n}}$.

The circuit $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{MJMedian}}}$ scales in the number of candidates and the number of grades since each grade needs a plaintext value per candidate, and the number of grades additionally increases the complexity of the computation of the median value per candidate. Figure 5.21 shows the constraints of the circuits. In this figure, we benchmark three variants: $n_{\mathsf{grades}} = 6$, $n_{\mathsf{grades}} = n_{\mathsf{cand}}$, and $n_{\mathsf{grades}} = \frac{n_{\mathsf{cand}}}{2}$. The figure shows that the circuit $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{MJMedian}}}$ is roughly split in half: one-half of the constraints is part of $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{MJMedian}}}$, while the other half is part of $\mathsf{Circ}^{\mathsf{Com}}$.

We present the benchmarks of the corresponding Groth16 SNARKs of $\mathsf{Circ}^{f^{\mathrm{res}}_{\mathrm{MJMedian}}}_{\mathsf{Com}}$ in Figure 5.24. The figure shows that the benchmarks scale linearly for a set of grades of constant size, while there is a quadratic scaling when the number of grades scales linearly in the number of candidates.

The benchmarks show that Kryvos efficiently evaluates $f^{\mathrm{res}}_{\mathrm{MJMedian}}$ for significant numbers of candidates and grades.

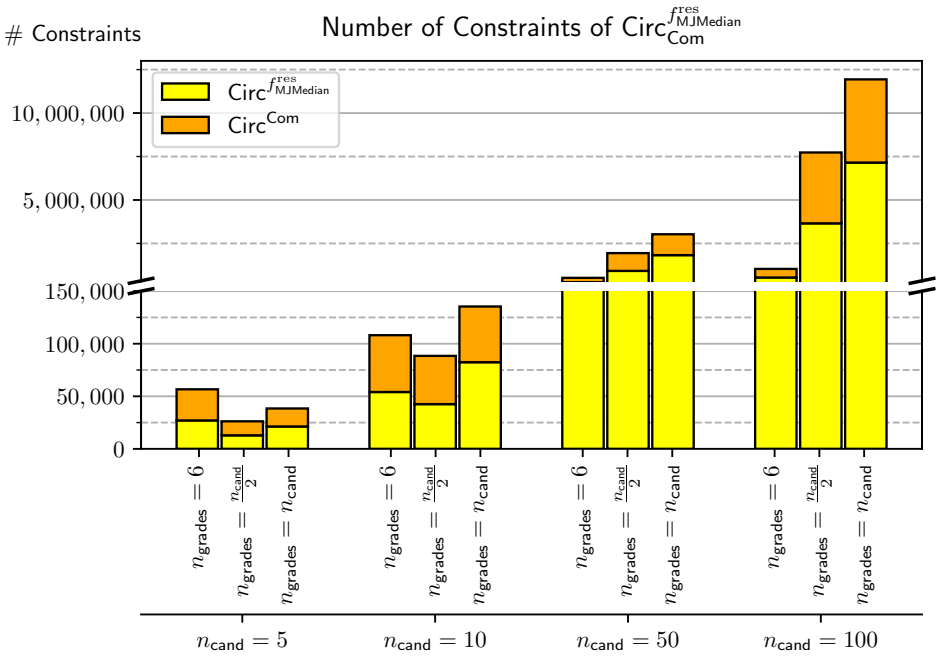


Figure 5.21.: Number of constraints of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}_{\mathsf{Com}}$.

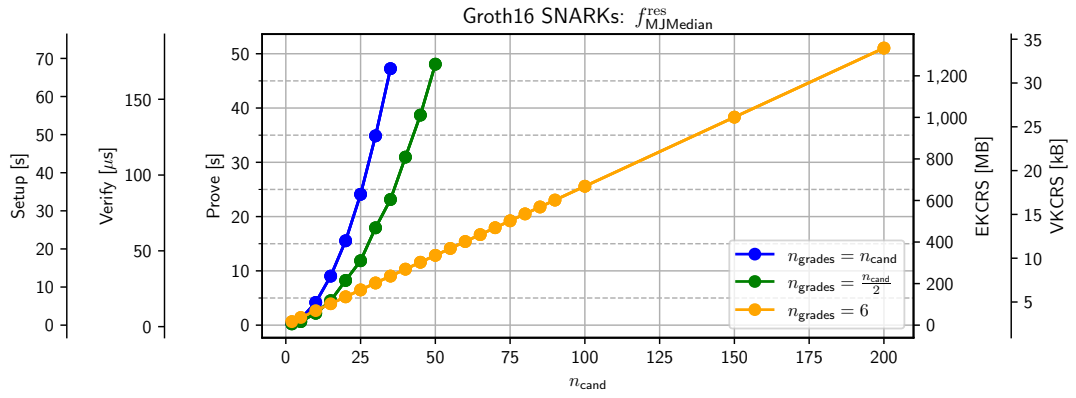


Figure 5.22.: Benchmarks of Groth16 SNARKs of $\Pi_{f^{\mathsf{res}}_{\mathsf{MJMedian}}}$.

5.3.5.15. Majority Judgment Full

The circuit $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}$ applies the optimized algorithm to compute the majority judgment method (see Section 2.7). Essentially, the circuit iterates over the candidates, using $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}$ as a sub-circuit, and eliminates one candidate for each iteration step. Therefore, on a high level, the complexity of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}$ corresponds to the complexity of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}$ multiplied by the number of candidates.

Figure 5.23 shows the constraints of the resulting circuits. Compared to $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}_{\mathsf{Com}}$, the sub-circuit $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}$ now dominates the overall circuit.

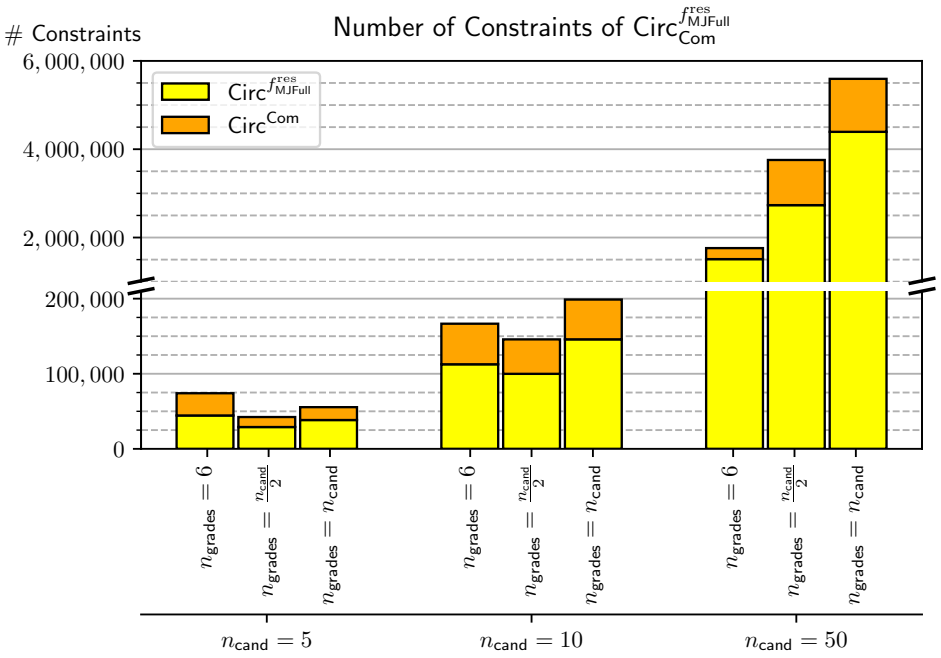


Figure 5.23.: Number of constraints of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}_{\mathsf{Com}}$.

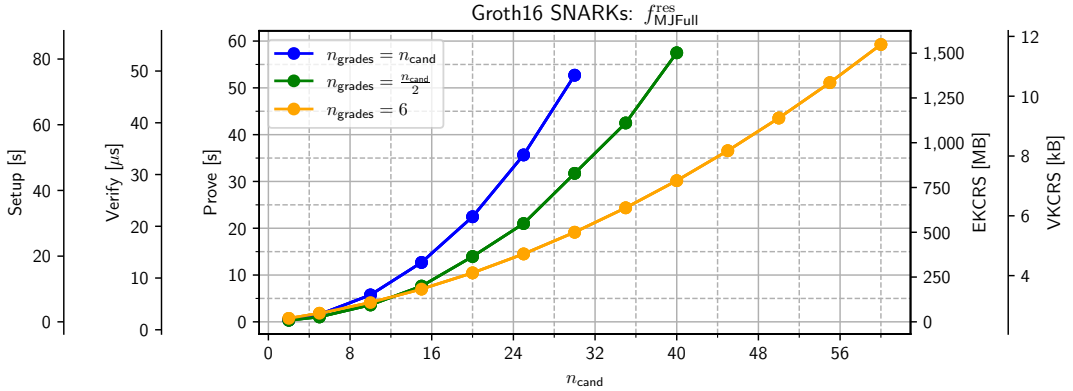


Figure 5.24.: Benchmarks of Groth16 SNARKs of $\Pi_{f^{\mathsf{res}}_{\mathsf{MJFull}}}$.

Figure 5.24 shows the benchmarks of the Groth16 SNARKs evaluating $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}_{\mathsf{Com}}$. The benchmarks indicate that Kryvos efficiently evaluates $f^{\mathsf{res}}_{\mathsf{MJFull}}$ for up to 20 candidates for $n_{\mathsf{grades}} = n_{\mathsf{cand}}$, and up to 50 candidates for $n_{\mathsf{grades}} = 6$.

5.3.5.16. Condorcet Smith

The circuit $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}$ computes the Smith set in two steps. First, it computes how many direct comparisons this candidate has won for each candidate. Then, the candidates that won the most direct comparisons are put into the current Smith set. The following theorem shows that this candidate is part of the Smith set.

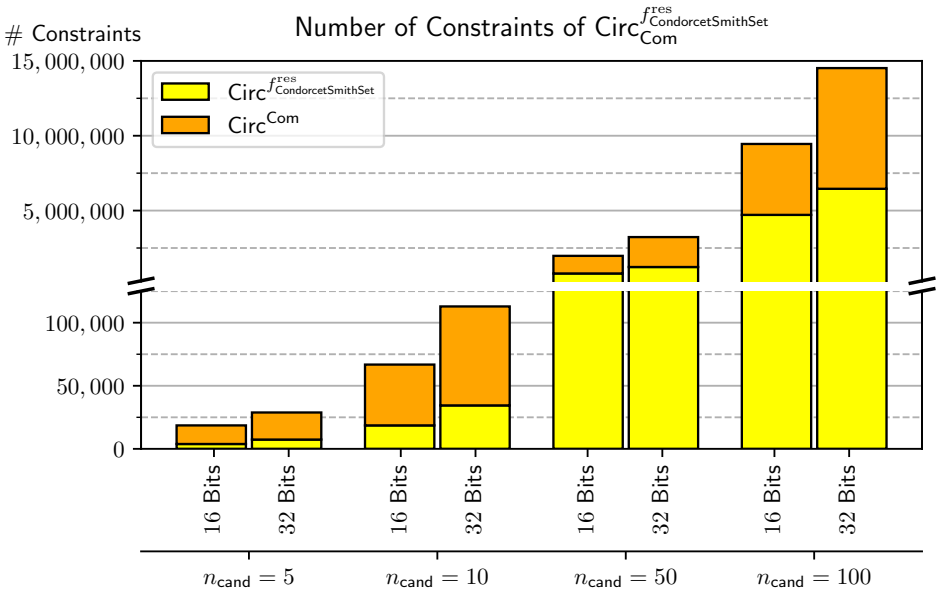


Figure 5.25.: Number of constraints of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}_{\mathsf{Com}}$.

Theorem 5.6 (Candidates with the most won duels are part of the Smith set)**.** *Let* $(c_i^{\mathsf{cand}})_{i=1}^{n_{\mathsf{cand}}}$ *be the candidates of an election with* $f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}$ *and let* $S \subseteq \{c_1^{\mathsf{cand}}, \ldots, c_{n_{\mathsf{cand}}}^{\mathsf{cand}}\}$ *be the Smith set. If* c_i^{cand} *has won the most duels, it holds that* $c_i^{\mathsf{cand}} \in S$*.*

Proof. Let $n = |S| \geq 1$, let c_i^{cand} be a candidate that won the most duels and let $c_i^{\mathsf{cand}} \notin S$. Since, by definition, no candidate not in the Smith set wins a duel against any candidate in the Smith set, c_i^{cand} can win at most $n_{\mathsf{cand}} - |S| - 1$ duels. However, a candidate $c_j^{\mathsf{cand}} \in S$ wins against every candidate not in the Smith set, i.e., she wins $n_{\mathsf{cand}} - |S| > n_{\mathsf{cand}} - |S| - 1$ many duels, and thus, c_i^{cand} did not win the most duels. □

Now, we have a set of candidates that are part of the Smith set by Theorem 5.6. In the second step, every candidate that wins a direct comparison against any candidate in the current Smith set is added to the set. By definition of the Smith set, such candidates are part of it. The algorithm iterates sufficiently many times until the Smith set is complete.

Figure 5.25 presents the constraint numbers of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}_{\mathsf{Com}}$. The figure shows that the constraints of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}$ make up about half of the constraints of $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}_{\mathsf{Com}}$ for larger numbers of candidates.

Figure 5.26 shows the benchmarks of the corresponding Groth16 SNARKs of $\Pi_{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}$. The benchmarks indicate that Kryvos can efficiently evaluate $f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}$ for 30 candidates.

5.3.5.17. Instant-Runoff Voting (IRV)

The circuits $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{IRVLotComplete}}}$, $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{IRVLotPartial}}}$, $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{IRVNSWCompelte}}}$, and $\mathsf{Circ}^{f^{\mathsf{res}}_{\mathsf{IRVNSWPartial}}}$ follow the same structure. They iterate over $n_{\mathsf{cand}} - 1$ round and eliminate one candidate per round. In each

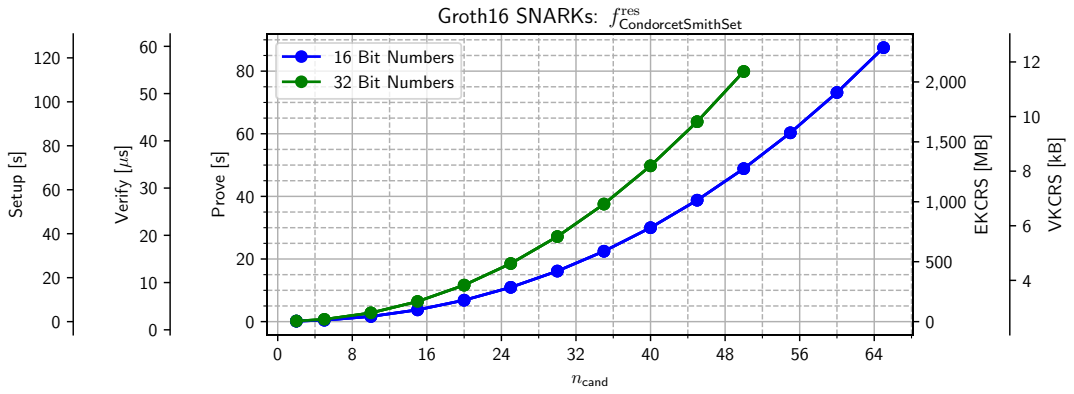


Figure 5.26.: Benchmarks of Groth16 SNARKs of $\Pi_{f^{\text{res}}_{\text{CondorcetSmithSet}}}$.

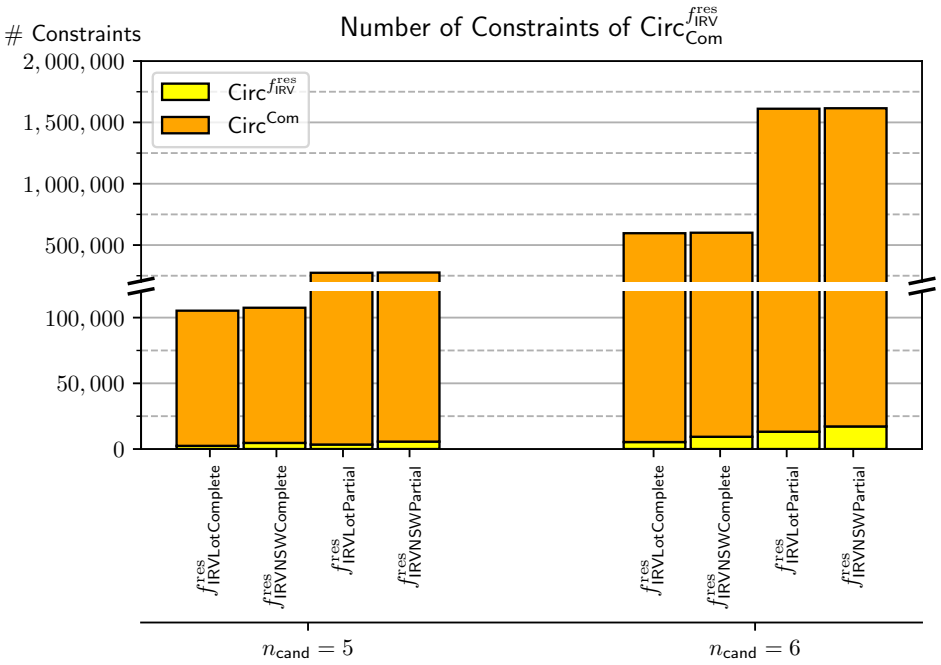


Figure 5.27.: Number of constraints of circuits for IRV.

round, the first votes per candidate are aggregated and compared, and tie-breaking is applied. The complexity of the tie-breaking mechanism scales in the number of candidates. In contrast, the complexity of Circ^{Com} scales exponentially in the number of candidates (see Section 2.7). We present the constraint numbers in Figure 5.27. These benchmarks indicate that the difference between the tie-breaking mechanisms is negligible, and the constraints of Circ^{Com} dominate the overall circuit.

Figure 5.28 presents the benchmarks of the corresponding Groth16 SNARKs, showing that Kryvos highly efficiently evaluates $\text{Circ}^{f^{\text{res}}_{\text{IRVLotComplete}}}$ and $\text{Circ}^{f^{\text{res}}_{\text{IRVLotPartial}}}$ for up to 5 candidates, and $\text{Circ}^{f^{\text{res}}_{\text{IRVNSWPartial}}}$ and $\text{Circ}^{f^{\text{res}}_{\text{IRVNSWCompelte}}}$ for up to 6 candidates. We note that the benchmarks of the different tie-breaking mechanisms are so close together that we merged them in Figure 5.28.

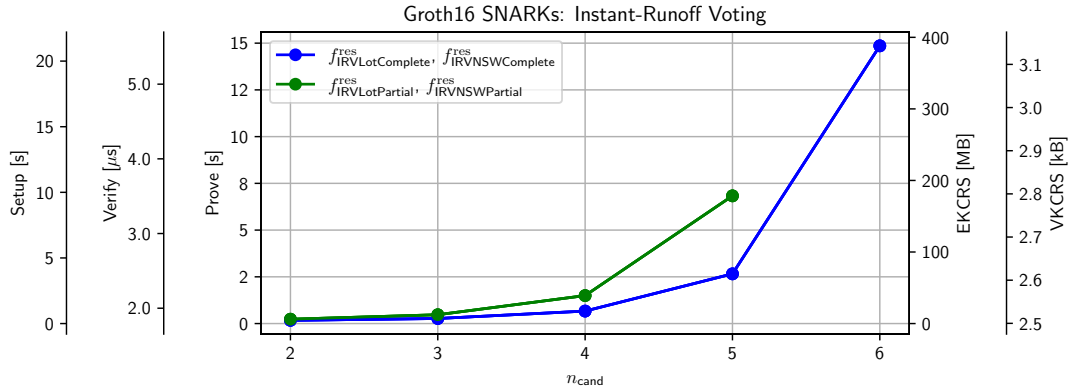


Figure 5.28.: Benchmarks of Groth16 SNARKs for IRV.

# Commitments ($= n_{\mathsf{tuples}}$)	1	2	5	10	50
Slot Size ($= N$)	1,957	979	392	196	40
Constraints	1,603,494	1,609,484	1,626,640	1,652,520	1,892,120
EKCRS [GB]	1.04	1.04	1.05	1.07	1.22
VKCRS [kB]	3.62	4.10	5.53	7.91	26.99
Prove [s]	39.66	39.81	40.24	40.88	46.80
Verify [ms]	8.27	11.03	19.30	33.08	143.34

Table 5.8.: Comparison of various slot sizes for $f^{\mathsf{res}}_{\mathsf{IRVNSWPartial}}$ with $n_{\mathsf{cand}} = 6$ and thus $n_{\mathsf{components}} = 1,957$.

Increasing the number of candidates no longer allows for efficient ballot submission. More precisely, voters cannot efficiently cast ballots that consist of a single commitment covering all (choice) components. As discussed in the case study for $\mathfrak{C}_{\mathsf{RankingPermutation}}$, we solve this issue by increasing n_{tuples}. Splitting the commitment affects the tallying SNARK since this SNARK must take the complete tally as input, which means it must verify n_{tuples} many commitments.

Tables 5.8 to 5.10 present benchmarks for various slot sizes for the next set of candidates. The benchmarks show that increasing n_{tuples} leads to slightly more constraints. However, having trustees with access to enough computational power (the main bottleneck is that they have at least 8 GB of RAM) allows the election to evaluate $f^{\mathsf{res}}_{\mathsf{IRVNSWPartial}}$ with $n_{\mathsf{cand}} = 7$. Verifying these SNARK proofs requires a VKCRS of small size (under 30 kB) and requires less than 200 ms.

In conclusion, efficient ballot submission is the main bottleneck for IRV. Efficient ballot submission is possible for complete rankings for up to 7 candidates and partial rankings for up to 6 candidates.

# Commitments ($= n_{\mathsf{tuples}}$)	1	2	5	10	50
Slot Size ($= N$)	5,040	2,520	1,008	504	101
Constraints	4,126,369	4,131,545	4,147,073	4,172,953	4,388,133
EKCRS [GB]	2.67	2.67	2.68	2.70	2.84
VKCRS [kB]	3.78	4.26	5.69	8.07	27.15
Prove [s]	102.07	102.20	102.58	103.22	108.55
Verify [ms]	9.19	11.94	20.21	34.00	144.26

Table 5.9.: Comparison of various slot sizes for $f^{\text{res}}_{\mathsf{IRVNSWCompelte}}$ with $n_{\mathsf{cand}} = 7$ and thus $n_{\mathsf{components}} = 5,040$.

# Commitments ($= n_{\mathsf{tuples}}$)	1	2	5	10	50
Slot Size ($= N$)	13,700	6,850	2,740	1,370	274
Constraints	11,244,153	11,249,329	11,264,857	11,290,737	11,497,777
EKCRS [GB]	7.27	7.27	7.28	7.30	7.43
VKCRS [kB]	3.78	4.26	5.69	8.07	27.15
Prove [s]	278.14	278.27	278.65	279.29	284.41
Verify [ms]	9.19	11.94	20.21	34.00	144.26

Table 5.10.: Comparison of various slot sizes for $f^{\text{res}}_{\mathsf{IRVNSWPartial}}$ with $n_{\mathsf{cand}} = 7$ and thus $n_{\mathsf{components}} = 13,700$.

5.3.5.18. Summary

With our numerous Kryvos instantiations, we efficiently support and implement several election result functions. The SNARK used to evaluate the election results function has a more relaxed runtime requirement than the SNARKs used for ballot creation. Users run these SNARKs on low-end devices, and the proof creation needs to be highly efficient. In contrast, a dedicated trustee who can use high-end machines for computation creates the SNARK proof for the election result function. Additionally, it is reasonable for the election evaluation to take longer than a few seconds, unlike ballot submission. We summarize the benchmarks Kryvos evaluating election result functions in Table 5.11 for various runtime limits. This table shows that Kryvos efficiently handles a variety of election result functions.

5.3.5.19. Public Verification Phase

Each Groth16 SNARK proof can be verified in $\sim$2.8 ms using the small VKCRS, where a single proof is of size $\sim$ 200 bytes. For example, consider a single-vote election ($\mathfrak{C}_{\mathsf{Single}}$) with an arbitrary number of trustees and up to $n_{\mathsf{components}} = 1,000$ candidates and $n_{\mathsf{voters}} = 100,000$ voters. Such an election requires 100,000 Groth16 SNARK proofs for showing the well-formedness of ballots

f^{res}	Parameters	n_{cand}		
		1 s	30 s	1 min
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Plurality}}}$	16 Bit Numbers	70	2,420	4,850
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Plurality}}}$	32 Bit Numbers	41	1,419	2,844
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$	16 Bit Numbers	53	1,846	3,700
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Threshold},t}}$	32 Bit Numbers	30	1,035	2,075
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Best},n}}$	16 Bit Numbers	42	1,454	2,016
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{Best},n}}$	32 Bit Numbers	23	795	1,594
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}$	$n_{\mathsf{grades}} = n_{\mathsf{cand}}$	4	27	39
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}$	$n_{\mathsf{grades}} = \frac{n_{\mathsf{cand}}}{2}$	6	39	55
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{MJMedian}}}$	$n_{\mathsf{grades}} = 6$	3	117	235
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}$	$n_{\mathsf{grades}} = n_{\mathsf{cand}}$	3	22	31
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}$	$n_{\mathsf{grades}} = \frac{n_{\mathsf{cand}}}{2}$	4	29	44
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{MJFull}}}$	$n_{\mathsf{grades}} = 6$	2	39	60
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}$	16 Bit Numbers	7	39	54
$\mathsf{Circ}_{\mathsf{Com}}^{f^{\mathsf{res}}_{\mathsf{CondorcetSmithSet}}}$	32 Bit Numbers	5	31	43

Table 5.11.: Maximum number of candidates for various election result functions that Kryvos can evaluate within a limited time. We present the results for IRV in Figure 5.28 and Tables 5.8 to 5.10.

and a single Groth16 SNARK proof for showing the correctness of the election result. In total, the size of all proofs is ∼20 MB with a total sequential verification time of ∼90 seconds. Verification requires two VKCRSs, both are smaller than 200 kB. This verification can be highly parallelized and already performed while other phases, such as the voting phase, are still running.

5.3.5.20. Real-World Elections with Kryvos

In this section, we instantiate the Kryvos framework for real-world elections.

House of Commons with Kryvos. The elections for the House of Commons use $\mathfrak{C}_{\mathsf{Single}}$ and $f^{\mathsf{res}}_{\mathsf{Plurality}}$ in each of the 650 constituencies. As presented above, Kryvos efficiently supports this choice space and election result function. Even for 1,500 candidates in a single constituency, voters can cast ballots for Kryvos in under 20 seconds with a EKCRS consisting of about 500

MB. For more realistic numbers of candidates below 250, voters need under 5 seconds to cast a ballot using an EKCRS consisting of under 100 MB. The trustees can evaluate $f^{\text{res}}_{\mathsf{Plurality}}$ efficiently for $3,600$ candidates, and parties can verify the election result under 5 ms using VKCRS of size under 1 MB. In summary, Kryvos efficiently supports the elections for the House of Commons.

Elections for the Fachkollegien in the Deutsche Forschungsgesellschaft with Kryvos. These elections use $f^{\text{res}}_{\mathsf{RankingVotesBest},n}$ with up to 32 candidates. Kryvos handles 32 candidates for this election result function in under five seconds for arbitrary numbers of voters.

Grand Final of the Eurovision Song Contest with Kryvos. Although the choice space of the ESC, namely $\mathfrak{C}_{\mathsf{BordaPointList}}$, is more complex than $\mathfrak{C}_{\mathsf{Single}}$, Kryvos efficiently supports this real-world election since there are few candidates and voters. Voters cast their ballots in under 2 seconds using a EKCRS of under 50 MB. As for the House of Commons, the trustees can efficiently compute and prove the election result.

Parliamentary Elections in the Republic of Nauru with Kryvos. To support the Dowdall system used by the parliamentary elections in the Republic of Nauru, we scale the points such that they are natural numbers. Since there are no more than 18 candidates, we scale the points by $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 17 = 510,510$, which means that a voter can give at most $510,510$ points to a candidate. Knowing that there are no more than $1,630$ voters in a constituency, a candidate can receive at most $832,131,300$ points, which we can represent with 32 bits. Therefore, we can efficiently evaluate the parliamentary elections in the Republic of Nauru with Kryvos.

The New South Wales Legislative Assembly with Kryvos. As discussed above, Kryvos efficiently evaluates $f^{\text{res}}_{\mathsf{IRVNSWPartial}}$ for up to 6 candidates. We run Kryvos by using real election data from the 2015 New South Wales state election for the Legislative Assembly [Ele15]. More specifically, we consider the electoral districts of Albury (five candidates) and Auburn (six candidates).

The Maine House of Representatives with Kryvos. Kryvos can efficiently handle elections with up to 6 candidates using IRV with $f^{\text{res}}_{\mathsf{IRVLotPartial}}$, making it suitable for the Maine House of Representatives, as well as the US presidential and senatorial elections in Maine.

5.4. Related Work

In this section, we compare Kryvos with related e-voting systems designed for tally-hiding. Furthermore, we discuss alternative design choices for Kryvos.

5.4.1. Comparison with Other Tally-Hiding E-Voting Systems

We compare Kryvos to existing tally-hiding e-voting systems in this section.

5.4.1.1. Fully Tally-Hiding E-Voting vs. Kryvos

We note that publicly tally-hiding and fully tally-hiding protocols provide the same level of privacy for the public. However, fully tally-hiding protocols keep the aggregated tally hidden from trustees, while publicly tally-hiding systems reveal the aggregated tally to the trustees to

allow for much higher efficiency, as demonstrated by Kryvos. The design of fully tally-hiding systems is fundamentally different to Kryvos as they are based on multi-party computation.

Canard et al. proposed a fully tally-hiding e-voting system [CPST18] specifically for majority judgment evaluation $f^{\text{res}}_{\mathsf{MJFull}}$ (see Section 2.7). However, there is a non-negligible chance that the system does not output a result because the underlying evaluation algorithm does not provide a result for every possible tally. Kryvos uses a different algorithm that always outputs the correct result. They implemented their system and provided benchmarks demonstrating that it can handle large numbers of voters, just like Kryvos. While their system needs almost 20 minutes to tally 5 candidates with 5 possible grades and up to $2^{20} - 1$ voters, Kryvos can handle all practically relevant numbers of candidates and grades with up to $2^{32} - 1$ voters, as shown in Figure 5.24. For example, Kryvos evaluated 10 candidates and 6 grades in under 5 seconds. Hence, Kryvos shows for the first time that publicly tally-hiding systems can not only realize majority judgment but also, as initially hoped, indeed achieve much better efficiency than a fully tally-hiding system (by providing weaker privacy towards trustees). Unlike Kryvos, [CPST18] lacks formal security analysis and was only benchmarked on a single computer, not a distributed network. Additionally, they used a simplified version of the cryptographic primitives for their benchmarks, and due to the online complexity of the underlying MPC protocol of [CPST18], it is unclear how well it performs in real-world distributed tallying scenarios.

Ordinos (presented in Section 3.2) is the first provably secure, verifiable, fully tally-hiding e-voting system. We instantiated and implemented it for manifold voting methods and result functions. The main difference between Kryvos and Ordinos is their balance between efficiency and the tally-hiding property they provide: Ordinos provides the stronger notion of *full* tally-hiding and takes several hours to evaluate complex election result functions with typical numbers of candidates (yet large numbers of voters), whereas Kryvos provides the relaxed notion of *public* tally-hiding and is practical even for very complex choice spaces (e.g., grading- and ranking-based choice spaces) and result functions (e.g., the Condorcet Smith set and IRV), taking under a single minute to evalute these elections for typical numbers of candidates. Unlike for Kryvos, the result function strongly impacts performance in Ordinos. The benchmarks of plurality voting of Ordinos provide a lower bound for all other result functions that Ordinos supports, which takes about an hour for 400 candidates in the fastest setting. In contrast, Kryvos evaluates $2,000$ candidates in about one minute for plurality voting in the fastest setting. Ordinos requires an hour for 20 candidates for the Condorcet Smith set ($f^{\text{res}}_{\mathsf{CondorcetSmithSet}}$), while Kryvos handles 48 candidates in about one minute. Furthermore, Kryvos provides efficient ZKPs showing ballot validity for all supported choice spaces. In contrast, Ordinos needs specialized ZKPs that might be inefficient for larger numbers of candidates.

5.4.1.2. Partially Tally-Hiding E-Voting vs. Kryvos

Several partially tally-hiding protocols have been proposed to solve the issue of Italian attacks in complex voting methods, such as Condorcet, Borda, IRV (e.g., [CM05, Hea07, BMN+09, JRRS19,

District	Albury		Auburn	
n_{cand}	5		6	
n_{voters}	46,347		43,783	
	Kryvos	[RCPT19]	Kryvos	[RCPT19]
Tallying	30 s	2h	46.80 s	15 h
Tallying: SNARK Prove [s]	14.8	N/A	46.80	N/A
Tallying: SNARK EKCRS [MB]	383	N/A	1,220	N/A
Tallying: SNARK VKCRS [kB]	3.13	N/A	26.99	N/A

Table 5.12.: Benchmarks of Kryvos for $f^{\mathsf{res}}_{\mathsf{IRVNSWPartial}}$ and of [RCPT19] for $f^{\mathsf{res}}_{\mathsf{IRVRoundTally}}$. We note that the benchmarks of [RCPT19] are taken directly from [RCPT19] which uses a setup comparable to the one we used for Kryvos. For Kryvos, we used a slot size of 40 for $n_{\mathsf{cand}} = 6$.

RCPT19]). All existing partially tally-hiding systems focus on mitigating Italian attacks, some of which are efficient and practical for real-world elections. The system of [RCPT19] is one of the most efficient systems evaluating $f^{\mathsf{res}}_{\mathsf{IRVRoundTally}}$ and has been shown to work for the 2015 New South Wales instant-runoff elections.

Partially and publicly tally-hiding systems offer different trade-offs between privacy and efficiency compared to fully tally-hiding ones. These trade-offs lead to incomparable privacy properties. Specifically, while publicly tally-hiding protocols hide the aggregated tally from the public, partially tally-hiding systems still reveal some intermediate information about the tally to the public. For example, in the voting protocol for instant-runoff elections by Ramchen et al. [RCPT19], the public learns, among others, the order of the weakest candidates, which might embarrass those candidates. Partially tally-hiding protocols hide parts of the tally even from internal parties, whereas publicly tally-hiding protocols reveal the aggregated tally to the trustees. As a result, Kryvos cannot protect against Italian attacks by the trustees, unlike the partially and fully tally-hiding systems mentioned earlier that are secure against such attacks.

We compare our $f^{\mathsf{res}}_{\mathsf{IRVNSWPartial}}$ evaluation benchmarks with the benchmarks of [RCPT19] in Table 5.12. We emphasize that the system of [RCPT19] evaluates $f^{\mathsf{res}}_{\mathsf{IRVRoundTally}}$ and therefore omits tie-breaking, as this can be done in plain. The benchmarks of [RCPT19] were obtained using a setup comparable to the one we used for Kryvos, consisting of an Intel i7-6770HQ, 2.6 GHz, with four cores (8 threads) and 32 GB RAM. However, we used a single core to obtain the benchmarks of Kryvos, while this information of the benchmarks of [RCPT19] is not specified. Note that the timings given for [RCPT19] are lower bounds since they exclude the runtime required for computing some of the NIZKPs during the tallying phase.

In terms of efficiency, Table 5.12 illustrates that Kryvos is well able to handle the same real-world IRV elections as [RCPT19]. Hence, both protocols are practical solutions for IRV elections that provide different incomparable balances in terms of privacy.

5.4.1.3. Italian Attacks

In this section, we elaborate on existing e-voting systems [CM05,Hea07,BMN+09,WB09,RCPT19, JRRS19] that aim to protect against Italian attacks.

Clarkson and Myers [CM05] proposed an extension of Prêt à Voter [CRS05] to protect against Italian attacks in Condorcet voting. This system comes with a quadratic overhead in the number of candidates. Neither formal security nor benchmarks are provided.

Heather [Hea07] published the first verifiable e-voting system that mitigates the risk of Italian attacks for single-transferable elections. Since the system reveals much sensitive information, this system, following a similar path, was first improved by Benaloh et al. [BMN+09], which in turn was improved by Ramchen et al. [RCPT19] using an MPC protocol that hides more information. In Table 5.12, we provide a performance comparison between Kryvos and [RCPT19], illustrating that Kryvos performs much better (while at the same time does not publicly reveal information).

The e-voting system by Wen and Buckland [WB09] for instant-runoff elections neither comes with a formal security analysis nor has this system been implemented. The theoretical complexity of [WB09] indicates that the resulting system may not be efficient.

Jamroga et al. [JRRS19] follow a completely different approach for reducing the amount of information revealed on individual voters' choices, inspired by risk-limiting auditing [LS12]. However, they describe their method only for majority voting.

5.4.2. Alternative Design Choices for Kryvos

This section discusses possible alternative design choices for Kryvos.

5.4.2.1. Specialized ZKPs for Ballot Validity for IRV

As described in Section 5.3.5, we use the choice space $\mathfrak{C}_{\mathsf{Single}}$ for IRV. This choice space allows specialized ZKPs to show ballot validity instead of relying on the SNARK proofs $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Single}}}$. In this section, we want to explore this possibility.

Specialized ZKPs for $\mathfrak{C}_{\mathsf{Single}}$ usually work as follows. The ZKPs consists of various sub-zero-knowledge proofs that allow for testing of one specific plaintext value of a ciphertext (or commitment):

$$R_{x,\mathsf{pk}}^{\text{value}} = \{(c, r) \mid c = \mathsf{Com}_{\mathsf{pk}}(x, r)\}$$

We combine these sub-ZKPs via an OR protocol. In the typical setting with $N = 1$, i.e., one commitment contains one value, we construct the ZKP as follows:

- For each commitment $c_i, i \in \{1, \ldots, n_{\mathsf{components}}\}$: $OR(c_i \in L_{R_{0,\mathsf{pk}}^{\text{value}}}, c_i \in L_{R_{1,\mathsf{pk}}^{\text{value}}})$
- For the aggregated commitment $c := \sum_{i \in \{1,\ldots,n_{\mathsf{components}}\}} : OR(c \in L_{R_{0,\mathsf{pk}}^{\text{value}}}, c \in L_{R_{1,\mathsf{pk}}^{\text{value}}})$

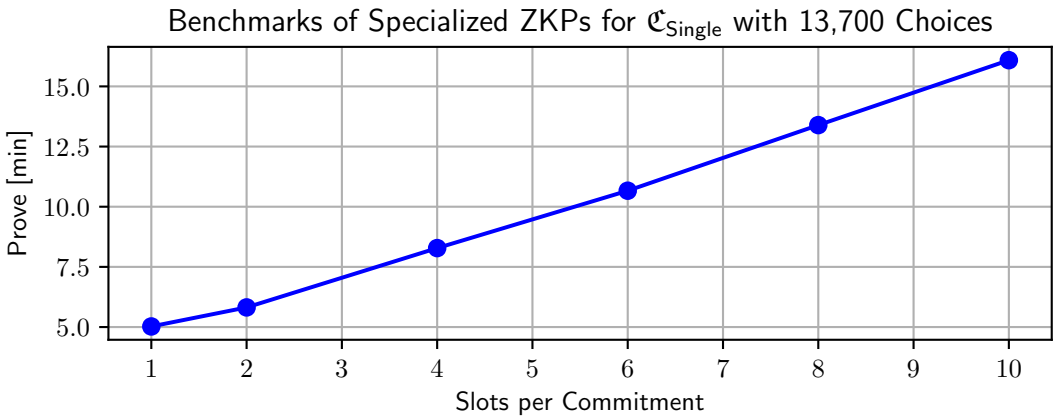


Figure 5.29.: Benchmarks Specialized ZKPs for $\mathfrak{C}_{\mathsf{Single}}$ with $13,700$ choices.

We can extend this construction to support Pedersen vector commitments. For this, the OR protocols have to test each possible combination of the values in the commitment. Therefore, the number of statements expressed inside each OR protocol increases while the overall number of OR protocols decreases. Additionally, increasing the slot size increases the cost of handling a commitment since it contains multiple values, and for each, we need to perform an exponentiation and multiplication.

We implemented such specialized ZKPs. Since the ZKPs $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Single}}}$ efficiently handle up to six candidates for a partial ranking, we provide benchmarks for seven candidates and a partial ranking, leading to $13,700$ choices for $\mathfrak{C}_{\mathsf{Single}}$ in Figure 5.29. The benchmarks show that increasing the slot size leads to more expensive ZKPs. Moreover, even for $N = 1$, creating a proof takes over five minutes, which is unsuitable for our e-voting setting. Therefore, we recommend using $\mathsf{Circ}_{\mathsf{Com}}^{\mathfrak{C}_{\mathsf{Single}}}$ to show ballot validity.

5.4.2.2. Distributed SNARK Proof Creation

It is possible to use the recent concept of distributed SNARKs as a potential solution to prove the correctness of the election result. This approach allows different parties, each holding a share of a witness, to compute a SNARK in a secure and distributed manner while preserving privacy [KZGM21, OB22]. However, this method has some downsides, considering tally-hiding e-voting. For instance, it relies heavily on MPC components to protect the witness shares' secrecy. It is unclear whether this approach would be more efficient than fully tally-hiding e-voting systems' MPC protocols, which cannot compete with Kryvos in terms of efficiency. According to benchmarks by [OB22], currently available distributed SNARKs are still too inefficient for our purposes.

Additionally, the computation of the distributed SNARK proof begins with the parties holding shares of the witness wires. However, for our voting circuits, the trustees only hold parts of the witness, the aggregated tally. The trustees must still compute the remaining parts of the

witness. If they do so in plain, we do not gain anything by employing distributed SNARKs. Therefore, we need to use MPC to securely compute the complete witness, increasing the overall computational overhead of distributed MPC.

5.4.2.3. CPSNARKs for Kryvos

We could employ commit-and-prove succinct non-interactive arguments of knowledge (CP-SNARKs) [CFQ19] in Kryvos. In a nutshell, CPSNARKs allow the prover to take plaintext values as input for the SNARK circuit and publish commitments to these values, such that one can verify the circuit on these commitments without knowing the plaintext values. Therefore, we can use CPSNARKs to decouple the commitments from the circuit, which would drastically reduce the number of constraints of the circuits employed in Kryvos, see Section 5.3.5.

However, assembling a secure e-voting system requires care, and not every proof system is directly applicable in this context. For example, Lee et al. [LCKO19] try to construct a secure e-voting system based on CPSNARKs, using Lego SNARKs [CFQ19]. Nevertheless, they face a severe issue with CPSNARKs: We can rerandomize the proofs together with the commitments of the inputs. Rerandomization allows the breaking of ballot privacy: Dishonest voters can rerandomize the proof and commitments of an honest voter and cast it as their ballot. Since CPSNARKs like Lego SNARK currently offer no countermeasures against this behavior, Lee et al. employ a robust blockchain in their voting protocol that mitigates these issues. However, a blockchain that satisfies the needed properties does not exist.

6. Conclusion

As presented in Chapter 1, the four main goals of this thesis are as follows.

1. We wanted to construct the first provable secure verifiable and full tally-hiding e-voting system supporting a wide range of election types.
2. We wanted to explore the impact of tally-hiding regarding the secrecy of the tally and confidentiality of individual votes.
3. We wanted to formalize the concept of public tally-hiding.
4. We wanted to construct the first provable secure verifiable and public tally-hiding e-voting system supporting a wide range of election types.

6.1. Secure Tally-Hiding E-Voting with Ordinos

We presented Ordinos in [KLM+20a], the first provably secure and verifiable fully tally-hiding remote e-voting protocol; it achieves verifiability and even accountability. Unlike previous protocols, Ordinos is a *modular framework* for fully tally-hiding e-voting that can be instantiated for supporting a wide range of election types, including essentially arbitrary voting methods. Our security analysis and proof are performed generically for the Ordinos framework itself and, therefore, carry over to all such instantiations.

We proposed and implemented various instantiations of Ordinos for many elections, including several complex Condorcet methods, IRV, and even the elections for the German Bundestag. Our extensive evaluations demonstrate that these instantiations achieve practical performance in everyday real-world settings. For this, we have tested our Ordinos instantiation for several real-world elections and the German Bundestag on the data from the 2021 election and demonstrated that we can perform even such a complex election on a real-world scale in a fully tally-hiding manner. This powerful insight highlights the potential for using Ordinos in significant and complex real-world elections.

Finally, we provided *a web framework of* Ordinos *that supports all steps of the voting process*, including setting up elections, the vote submission (including voter authentication), the secure tally-hiding evaluation of the election result, and all verification steps necessary to verify the result of the election. The web framework fully supports all our Ordinos instantiations and voting methods, except for the German Bundestag elections.

Applying the Ordinos framework, we proposed several partial tally-hiding Ordinos instantiations and analyzed their trade-off between privacy and efficiency in comparison to their corresponding full tally-hiding Ordinos instantiations, showing a massive improvement in terms of efficiency.

6.2. Analysing the Impact of Tally-Hiding

We provided insights into the impact of tally-hiding by analyzing and comparing the privacy in different elections with and without tally-hiding. This includes relatively simple single-vote elections but also very challenging and complex functions, e.g., several Condorcet methods and IRV. Our analysis shows that tally-hiding severely improves privacy, particularly for elections with complex and large choice spaces, such as ranking-based voting methods like Condorcet and IRV.

Furthermore, we explored the impact of tally-hiding when applying the partial tally-hiding result functions from our partial tally-hiding Ordinos instantiations. Our findings show that revealing some information besides the election result allows for drastically improving performance while at the same time obtaining similar levels of privacy in comparison to full tally-hiding.

6.3. Formalizing the Concept of Public Tally-Hiding

Our research focused on the development of *publicly tally-hiding* techniques for e-voting systems. These techniques differ from previous methods and offer a new balance between privacy and efficiency.

As verifiable publicly tally-hiding e-voting is novel, we created a general definitional security model. This model provides the first formal definition of the public tally-hiding property. It enables the formal and rigorous modeling of publicly tally-hiding e-voting systems and their security properties. This includes the ability to ensure vote privacy and verifiability.

6.4. Secure Tally-Hiding E-Voting with Kryvos

We have introduced the Kryvos framework, which is the world's first verifiable publicly tally-hiding e-voting system that is provably secure. The Kryvos framework is a flexible voting system that can be customized for different elections by designing or fixing appropriate cryptographic building blocks, similar to Ordinos. Our design choices for Kryvos are novel, and we utilize efficient zero-knowledge proofs to create an efficient system that achieves practical performance for various real-world elections.

To evaluate elections using Kryvos, we have developed and implemented optimized versions of the advanced Groth16 SNARK [Gro16] for a wide range of voting methods. We have created instantiations of Kryvos for many commonly used voting methods, from simple to complex. Through thorough performance testing, we have shown that our publicly tally-hiding Kryvos framework performs significantly better than Ordinos instances. Specifically, for highly complex

voting methods such as Condorcet and IRV, our Kryvos instances support more candidates than Ordinos instances. Additionally, our Kryvos scheme outperforms even the state-of-the-art IRV partial tally-hiding e-voting scheme of Ramchen et al. [RCPT19], while achieving incompatible privacy properties.

As a significant side product, we constructed highly efficient zero-knowledge proofs (ZKPs) showing ballot validity for many different choice spaces. We do not only construct ZKPs for relatively simple choice spaces like single- and multi-vote, but most impactfully for very complex choice spaces, for which no ZKPs exist so far. For instance, we are the first to propose ZKPs for the Borda tournament style choice space. Our constructions offer high performance, allowing voters to prove the validity of their ballots very efficiently: Voters can prove in under a single second ballot validity of single-choice for up to 73 candidates. Furthermore, voters can prove ballot validity for complex choice spaces in under one second for up to 32 candidates in the case of Borda tournament style and up to 9 candidates for Condorcet methods. Our study of real-world elections shows that these are realistic numbers, thus allowing us to prove ballot validity for real-world elections very efficiently. Moreover, our ZKP constructions are not limited to public tally-hiding. They can be used in other e-voting schemes, providing a new foundation for efficiently handling simple and complex choice spaces.

A. Cryptographic Primitives

This section provides definitions and security notions for various cryptographic primitives used in this thesis. We present threshold homomorphic encryption in Appendix A.1, digital signature in Appendix A.2, bilinear groups in Appendix A.3 and non-interactive zero-knowledge proofs (NIZKPs) in Appendix A.4.

A.1. Threshold Homomorphic Encryption

In this section, we present threshold homomorphic encryption. We define such schemes in Appendix A.1.1 and the corresponding security notion in Appendix A.1.2.

A.1.1. Threshold Public-Key Encryption Scheme

Let n_{trustees} be the number of trustees T_k and t be a threshold. Let prm be the parameters, including the security parameter 1^η. We implicitly assume that all algorithms have prm as input. A $(n_{\mathsf{trustees}}, t)$-*threshold public-key encryption scheme* is a tuple of polynomial-time algorithms $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, E_{\mathsf{pk}}, \mathsf{dec}^{\mathsf{share}}, \mathsf{dec})$ such that we have:

- $\mathsf{KeyShareGen}$ (which is run by a single trustee T_k) is probabilistic and outputs two keys $(\mathsf{pk}_k, \mathsf{sk}_k)$, called the *public-key share* pk_k and the *secret-key share* sk_k,
- $\mathsf{PublicKeyGen}$ is deterministic and takes as input n_{trustees} public-key shares $\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{trustees}}}$, and outputs a public key pk; this algorithm may fail (output $\perp$) if the public-key shares are invalid,
- E_{pk} is probabilistic and takes as input a public key pk and a message a, and outputs a ciphertext $E_{\mathsf{pk}}(a)$,
- $\mathsf{dec}^{\mathsf{share}}$ (which is run by a single trustee T_k) is probabilistic and takes as input a ciphertext $E_{\mathsf{pk}}(a)$ and a secret-key share sk_k, and outputs a decryption share $\mathsf{dec}^{\mathsf{share}}_{k,E_{\mathsf{pk}}(a)}$,
- dec is deterministic and takes as input a tuple of decryption shares of size at least t and returns a message a or $\perp$, in the case that decryption fails.

Furthermore, the following correctness condition has to be guaranteed. Let $(\mathsf{pk}_k, \mathsf{sk}_k)$ be generated by $\mathsf{KeyShareGen}$ for all $k \in \{1, \ldots, n_{\mathsf{trustees}}\}$ and let pk be generated by the key generation algorithm $\mathsf{PublicKeyGen}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{trustees}}})$. Let $E_{\mathsf{pk}}(a)$ be the ciphertext obtained by

encrypting a under the public key pk and $\mathsf{dec}^{\mathsf{share}}_{k,E_{\mathsf{pk}}(a)}$ be an output of $\mathsf{dec}^{\mathsf{share}}_{E_{\mathsf{pk}}(a)}(sk_k)$ for $k \in I$, where $I \subseteq \{1, \ldots, n_{\mathsf{trustees}}\}$. Then, we have

$$\mathsf{dec}((\mathsf{dec}^{\mathsf{share}}_{k,E_{\mathsf{pk}}(a)})_{k\in I}) = \begin{cases} a & \text{if } |I| \geq t \\ \bot & \text{otherwise} \end{cases}.$$

A.1.2. IND-CPA Security

Let $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, E_{\mathsf{pk}}, \mathsf{dec}^{\mathsf{share}}, \mathsf{dec})$ be a $(n_{\mathsf{trustees}}, t)$-threshold public-key encryption scheme.

Let $\mathsf{Ch}^{E_{\mathsf{pk}}}$ be a ppt algorithm, called a *challenger*, which takes as input a bit b and a public key pk and serves the following challenge queries: For a pair of messages (a_0, a_1) of the same length, return $E_{\mathsf{pk}}(b)$ if $\mathsf{pk} \neq \bot$, or $\bot$ otherwise.

Let $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$ be an adversary, where $\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3$ share state and A_3 has oracle access to $\mathsf{Ch}^{E_{\mathsf{pk}}}$.

Let $\mathsf{Exp}_{\mathsf{A}}(b)$ be defined as follows:

1. $I \leftarrow \mathsf{A}_1()$ where $I \subseteq \{1, \ldots, n_{\mathsf{trustees}}\}$ and $|I| \geq t$
2. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyShareGen}()$ for $i \in I$
3. $\mathsf{pk}_j \leftarrow \mathsf{A}_2(\{\mathsf{pk}_i\}_{i\in I})$ for $j \in \{1, \ldots, n_{\mathsf{trustees}}\} \setminus I$
4. $\mathsf{pk} \leftarrow \mathsf{PublicKeyGen}(\mathsf{pk}_1, \ldots, \mathsf{pk}_{n_{\mathsf{trustees}}})$
5. $b' \leftarrow \mathsf{A}_3^{\mathsf{Ch}^{E_{\mathsf{pk}}}(b,\mathsf{pk})}()$
6. output b'

We define that the $(n_{\mathsf{trustees}}, t)$-threshold public-key encryption scheme is *IND-CPA secure* if for all (polynomially bounded) adversaries $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2, \mathsf{A}_3)$

$$\Pr(\mathsf{Exp}_{\mathsf{A}}(0) \text{ outputs } 1) - \Pr(\mathsf{Exp}_{\mathsf{A}}(1) \text{ outputs } 1)$$

is negligible as a function in the security parameter η.

A.2. Digital Signatures

In this section, we present signature schemes. We define such schemes in Appendix A.2.1 and the corresponding security notion in Appendix A.2.2.

A.2.1. Signature Schemes

A *digital signature scheme* consists of a triple of algorithms $(\mathsf{KeyGen}, \mathsf{sign}, \mathsf{verify})$, where

1. KeyGen, the *key generation algorithm*, is a probabilistic algorithm that takes a security parameter η and returns a pair $(\mathsf{sign}_i, \mathsf{verify}_i)$ of matching secret signing and public verification keys.

2. sign, the *signing algorithm*, is a (possibly) probabilistic algorithm that takes a private signing key sign_i and a message $\mathsf{m} \in \{0,1\}^*$ to produce a signature $\mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m})$.

3. verify, the *verification algorithm*, is a deterministic algorithm which takes a public verification key verify_i, a message $\mathsf{m} \in \{0,1\}^*$ and a signature $\mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m})$ to produce a boolean value.

We require that for all key pairs $(\mathsf{sign}_i, \mathsf{verify}_i)$ which can be output by $\mathsf{KeyGen}(1^\eta)$, for all messages $\mathsf{m} \in \{0,1\}^*$, and for all signatures $\mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m})$ that can be output by $\mathsf{sign}(\mathsf{sign}_i, \mathsf{m})$, we have that $\mathsf{verify}(\mathsf{verify}_i, \mathsf{m}, \mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m})) = \text{true}$. We also require that KeyGen, sign, and verify can be computed in polynomial time.

A.2.2. EUF-CMA-Security

Let $\mathcal{S} = (\mathsf{KeyGen}, \mathsf{sign}, \mathsf{verify})$ be a signature scheme with security parameter η. Then $\mathcal{S}$ is *existentially unforgeable under adaptive chosen-message attacks (EUF-CMA-secure)* if for every probabilistic (polynomial-time) algorithm A who has access to a signing oracle and who never outputs tuples $(\mathsf{m}, \mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m}))$ for which m has previously been signed by the oracle, we have that

$$\begin{aligned}\Pr((\mathsf{sign}_i, \mathsf{verify}_i) \leftarrow \mathsf{KeyGen}(1^\eta);\\ (\mathsf{m}, \mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m})) \leftarrow \mathsf{A}^{\mathsf{sign}(\mathsf{sign}_i,\cdot)}(1^\eta, \mathsf{verify}_i);\\ \mathsf{verify}(\mathsf{verify}_i, \mathsf{m}, \mathsf{sig}_{\mathsf{sign}_i}(\mathsf{m})) = \text{true}\end{aligned}$$

is negligible as a function in η.

A.3. Bilinear Groups

Bilinear groups can be used as a building block for publicly verifiable SNARGs and SNARKs. They are, for example, used in the Groth16 SNARK [Gro16].

Definition A.1 (Bilinear Group [Gro16])**.** *A bilinear group is a tuple* $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ *with the following properties:*

- $\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T$ *are groups of prime order* p*.*

- $e : \mathbb{G}_1, \mathbb{G}_2 \times \mathbb{G}_T$ *is a bilinear map, i.e.*
  - $\forall u \in \mathbb{G}_1, v \in \mathbb{G}_2, a, b \in \mathbb{Z}_p : e(u^a, v^b) = e(u, v)^{ab}$
  - $\forall u, v \in \mathbb{G}_1, w \in \mathbb{G}_2 : e(uv, w) = e(u, w) \cdot e(v, w)$
- *If* G *is a generator for* $\mathbb{G}_1$ *and* H *is a generator for* $\mathbb{G}_2$ *then* $e(G, H)$ *is a generator for* $\mathbb{G}_T$.
- *There are efficient algorithms for computing group operations, evaluating the bilinear map, deciding group membership, deciding the equality of group elements, and sampling generators of the groups.*

A.4. Non-Interactive Zero-Knowledge Proofs

In this section, we present non-interactive zero-knowledge proofs (NIZKPs). We define such schemes in Appendix A.4.1, including the corresponding security notions. Then, we present the NIZKPs employed in Ordinos in Appendix A.4.2. Finally, in Appendix A.4.3, we present the Groth16 SNARK from [Gro16].

A.4.1. Definitions

Non-Interactive Proof Systems. Let $\mathcal{R}$ be an efficiently computable binary relation. For pairs $(x, w) \in \mathcal{R}$, x is called the *statement* and w is called the *witness*. Let $L_{\mathcal{R}} = \{x \mid \exists w : (x, w) \in \mathcal{R}\}$. A *non-interactive proof system* for the language $L_{\mathcal{R}}$ is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$, where

- Setup (the CRS generator) takes as input a security parameter 1^η and the statement length n and produces a common reference string $\sigma \leftarrow \mathsf{Setup}(n)$, For simplicity of notation, we omit the security parameter in the notation, also for the prover and the verifier.
- Prove takes as input the security parameter 1^η, a common reference string σ, a statement x, and a witness w and produces a proof $\pi \leftarrow \mathsf{Prove}(\sigma, x, w)$,
- Verify takes as input the security parameter 1^η, a common reference string σ, a statement x, and a proof π and outputs $1/0 \leftarrow \mathsf{Verify}(\sigma, x, w)$ depending on whether it accepts π as a proof of x or not,

such that the following conditions are satisfied:

- *(Computational) Completeness:* Let $n = \eta^{O(1)}$ and A be an adversary that outputs $(x, w) \in \mathcal{R}$ with $|x| = n$. Then, the probability

$$\Pr(\sigma \leftarrow \mathsf{Setup}(n); (x, w) \leftarrow \mathsf{A}(\sigma); \\ \pi \leftarrow \mathsf{Prove}(\sigma, x, w); b \leftarrow \mathsf{Verify}(\sigma, x, \pi) : b = 1)$$

is overwhelming (as a function of the security parameter 1^η). In other words, this condition guarantees that an honest prover should always be able to convince an honest verifier of a true statement (which can be chosen by the adversary A).

- *(Computational) Soundness:* Let $n = \eta^{O(1)}$ and A be a non-uniform polynomial time adversary. Then, the probability

$$\begin{aligned}&\Pr(\sigma \leftarrow \mathsf{Setup}(n); (x, \pi) \leftarrow \mathsf{A}(\sigma);\\ &\quad b \leftarrow \mathsf{Verify}(\sigma, x, \pi) \colon b = 1 \text{ and } x \notin L_{\mathcal{R}})\end{aligned}$$

  is negligible (as a function of the security parameter 1^η). In other words, this condition guarantees that it should be infeasible for an adversary to come up with a proof π of a false statement x that the verifier accepts.

Zero-Knowledge. We define that a non-interactive proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ is *zero-knowledge* (NIZKP) if the following condition is satisfied.

Let $n = \eta^{O(1)}$. There exists a polynomial-time simulator $\mathrm{Sim} = (\mathrm{Sim}_1, \mathrm{Sim}_2)$ such that for all stateful, interactive, non-uniform polynomial-time adversaries $\mathsf{A} = (\mathsf{A}_1, \mathsf{A}_2)$ that output $(x, w) \in \mathcal{R}$ with $|x| = n$, we have

$$\begin{aligned}&\Pr(\sigma \leftarrow \mathsf{Setup}(n); (x, w) \leftarrow \mathsf{A}_1(\sigma);\\ &\quad \pi \leftarrow \mathsf{Prove}(\sigma, x, w); b \leftarrow \mathsf{A}_2(\pi) \colon b = 1)\\ \approx\ &\Pr((\sigma, \tau) \leftarrow \mathrm{Sim}_1(n); (x, w) \leftarrow \mathsf{A}_1(\sigma);\\ &\quad \pi \leftarrow \mathrm{Sim}_2(\sigma, x, \tau); b \leftarrow \mathsf{A}_2(\pi) \colon b = 1)\end{aligned}$$

(where $\approx$ means that the difference between the two probabilities is negligible as a function of the security parameter).

We are using the *single-theorem variant* of the zero-knowledge property for our purposes, meaning that the CRS is only used to create and verify one ZK proof rather than many. This is sufficient for Ordinos since the number of NIZKP proofs is limited. This limitation corresponds to a scenario where the adversary can only submit a restricted number of queries. In this case, the single-theorem variant of the zero-knowledge property is enough to imply the multi-theorem variant. To expand the length of σ, which is bounded by the number of NIZKPs, a factor of M can be used, where M is the bound on the number of NIZKPs.

Proof of Knowledge. We define that a non-interactive proof system $(\mathsf{Setup}, \mathsf{Prove}, \mathsf{Verify})$ produces a *proof of knowledge* if the following condition is satisfied.

There exists a *knowledge extractor* $\mathrm{Extractor} = (\mathrm{Extractor}_1, \mathrm{Extractor}_2)$ such that for $n = \eta^{O(1)}$, the following conditions hold true:

- For all non-uniform polynomial-time adversaries A, we have that

$$\begin{aligned}&\Pr(\sigma \leftarrow \mathsf{Setup}(n); b \leftarrow \mathsf{A}(\sigma) \colon b = 1)\\ \approx\ &\Pr((\sigma, \tau) \leftarrow \mathrm{Extractor}_1(n); b \leftarrow \mathsf{A}(\sigma) \colon b = 1).\end{aligned}$$

- For all non-uniform polynomial-time adversaries A, we have that the probability

$$\begin{aligned}\Pr((\sigma, \tau) \leftarrow \mathrm{Extractor}_1(n); (x, \pi) \leftarrow \mathsf{A}(\sigma);\\ w \leftarrow \mathrm{Extractor}_2(\sigma, \tau, x, \pi);\\ b \leftarrow \mathsf{Verify}(\sigma, x, \pi) \colon b = 0 \text{ or } (x, w) \in \mathcal{R})\end{aligned}$$

  is overwhelming (as a function of the security parameter).

Note that (computational) knowledge extraction implies the existence of a witness and, therefore, it implies (computational) adaptive soundness.

A.4.2. (NIZK) Proofs used in Ordinos

Let $\mathcal{E} = (\mathsf{KeyShareGen}, \mathsf{PublicKeyGen}, E_{\mathsf{pk}}, \mathsf{dec}^{\mathsf{share}}, \mathsf{dec})$ be a (threshold) public-key encryption scheme. Then, the zero-knowledge proofs used in the voting protocol are formally defined as follows:

- *NIZKP $\pi^{\mathsf{KeyShareGen}}$ of knowledge and correctness of the private key share.* For a given public key pk_i, the statement is:

$$\exists \mathsf{sk}_i\ \exists r \colon (\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{KeyShareGen}(r)$$

- *NIZKP $\pi^{E_{\mathsf{pk}}()}$ of knowledge and correctness of plaintext(s).* Let $\mathcal{R}_x$ be an n-ary relation over the plaintext space. For $(E_{\mathsf{pk}}(x_1), \ldots, E_{\mathsf{pk}}(x_n), \mathsf{pk})$, the statement is:

$$\exists (x_1, \ldots, x_n) \in \mathcal{R}_x\ \forall i\ \exists r_i \colon E_{\mathsf{pk}}(x_i) = E_{\mathsf{pk}}(\mathsf{pk}, x_i; r_i).$$

A.4.3. The Groth16 SNARK

This section briefly presents the Groth16 SNARK proposed in [Gro16]. This SNARK is a linear interactive proof (see [Gro16]) that is run over bilinear groups $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e)$ (see Appendix A.3).

The relation of the Groth16 SNARK is of the form

Groth16 SNARK

- $\mathsf{Setup}(\mathcal{R})$: (CRS, τ)
  1. Pick arbitrary generators G and H for $\mathbb{G}_1$ and $\mathbb{G}_2$.
  2. $\alpha, \beta, \gamma, \delta, x \xleftarrow{\$} \mathbb{Z}_p^*$, $\tau = (\alpha, \beta, \gamma, \delta)$.
  3. $$\mathsf{CRS} = \left(G^{\alpha}, G^{\beta}, H^{\beta}, H^{\gamma}, G^{\delta}, H^{\delta}, \{G^{x}\}_{i=0}^{n-1}, \left\{ G^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}} \right\}_{i=0}^{l}, \right.$$ $$\left. \left\{ G^{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}} \right\}_{i=l+1}^{m}, \left\{ G^{\frac{x^i t(x)}{\delta}} \right\}_{i=0}^{n-2} \right)$$
  4. **output** (CRS, τ).
- $\mathrm{Prove}(\mathcal{R}, \mathsf{CRS}, a_1, \ldots, a_m)$: π
  1. Pick $r, s \xleftarrow{\$} \mathbb{Z}_p$ and compute $\pi = (A, B, C)$, where
     $$A = G^{\alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta}, B = H^{\delta + \sum_{i=0}^{m} a_i v_i(x) + s\delta},$$
     $$C = G^{\frac{\sum_{i=l+1}^{m} a_i (\beta u_i(x) + \alpha v_i(x) + w_i(x)) + h(x)t(x)}{\delta} + s(\alpha + \sum_{i=0}^{m} \alpha_i u_i(x)) + r(\beta + \sum_{i=0}^{m} a_i v_i(x)) + rs\delta}$$
  2. **output** π.
- $\mathrm{Vfy}(\mathcal{R}, \mathsf{CRS}, a_1, \ldots, a_l, \pi)$: 0/1
  1. Parse $\pi = (A, B, C) \in \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_1$.
  2. Accept the proof if and only if
     $$e(A, B) = e(G^{\alpha}, H^{\beta}) e(G^{\frac{\sum_{i=0}^{l} a_i (\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\gamma}}, H^{\gamma}) e(C, H^{\delta})$$

Figure A.1.: Groth16 SNARK [Gro16].

$$\mathcal{R} = (p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, e, l, \{u_i(X), v_i(X), w_i(X)\}_{i=0}^{m}, t(X)),$$

with $|p| = \lambda$. The relation defines a field $\mathbb{Z}_p$ and a language of statements $(a_1, \ldots, a_l) \in \mathbb{Z}_p^l$ and witnesses $(a_{l+1}, \ldots, a_m) \in \mathbb{Z}_p^{m-l}$ such that with $a_0 = 1$

$$\sum_{i=0}^{m} a_i u_i(X) \cdot \sum_{i=0}^{m} a_i v_i(X) = \sum_{i=0}^{m} a_i w_i(X) + h(X)t(X)$$

for some degree $n-2$ quotient polynomial $h(X)$, where n is the degree of $t(X)$.

The work of [Gro16] presents the standard method to transform a QAP into this relation.

Groth16 SNARK: Simulator

- $\text{Sim}(\mathcal{R}, \tau, a_1, \ldots, a_l)$: π
  1. Pick $r, s \xleftarrow{\$} \mathbb{Z}_p$ and compute a simulated proof $\pi = (A, B, C)$ as
  $$A = G^r, B = H^s, C = G^{\frac{rs - \alpha\beta - \sum_{i=0}^{l} a_i(\beta u_i(x) + \alpha v_i(x) + w_i(x))}{\delta}}$$
  2. **output** π.

Figure A.2.: Simulator for the Groth16 SNARK [Gro16].

We present the SNARK in Figure A.1 and the simulator in Figure A.2.

In the following, we prove the correctness of the linear interactive proof of the Groth16 SNARK. This linear interactive proof differs from the Groth16 SNARK in that it does not use bilinear groups but is instead restricted to linear provers (see [Gro16] for more details).

Lemma A.1 (Completeness of the Groth16 SNARK)**.** *The underlying linear interactive proof of the Groth16 SNARK presented in Figure A.1 fulfills completeness.*

Proof. Let A, B, C as defined in the description of the linear interactive proof:

$$\begin{aligned}
A &= \alpha + \sum_{i=0}^{m} a_i u_i(x) + r\delta \\
B &= \beta + \sum_{i=0}^{m} a_i v_i(x) + s\delta \\
C &= \frac{\sum_{i=l+1}^{m} a_i \left(\beta u_i(x) + \alpha v_i(x) + w_i(x)\right) + h(x)t(x)}{\delta} + As + rB - rs\delta
\end{aligned}$$

Then, the following holds.

$$\begin{aligned}
A \cdot B = \alpha \cdot \beta &+ \alpha \sum_{i=0}^{m} a_i v_i(x) + \alpha s\delta \\
&+ \beta \sum_{i=0}^{m} a_i u_i(x) + \underbrace{\sum_{i=0}^{m} a_i u_i(x) \cdot \sum_{i=0}^{m} a_i v_i(x)}_{= \sum_{i=0}^{m} a_i w_i(x) + h(x)t(x)} + s\delta \sum_{i=0}^{m} a_i u_i(x) \\
&+ r\delta\beta + r\delta \sum_{i=0}^{m} a_i v_i(x) + rs\delta^2
\end{aligned}$$

We rearrange the terms:

$$= \alpha \cdot \beta + \alpha \sum_{i=0}^{m} a_i v_i(x) + \beta \sum_{i=0}^{m} a_i u_i(x) + \sum_{i=0}^{m} a_i w_i(x) + h(x)t(x)$$
$$+ \delta \left(s \sum_{i=0}^{m} a_i u_i(x) + r \sum_{i=0}^{m} a_i v_i(x) + \alpha s + r\beta + rs\delta \right)$$

We split the sums:

$$= \alpha \cdot \beta + \alpha \sum_{i=0}^{l} a_i v_i(x) + \beta \sum_{i=0}^{l} a_i u_i(x) + \sum_{i=0}^{l} a_i w_i(x)$$
$$+ \alpha \sum_{i=l+1}^{m} a_i v_i(x) + \beta \sum_{i=l+1}^{m} a_i u_i(x) + \sum_{i=l+1}^{m} a_i w_i(x) + h(x)t(x)$$
$$+ \delta \left(s \sum_{i=0}^{m} a_i u_i(x) + r \sum_{i=0}^{m} a_i v_i(x) + \alpha s + r\beta + rs\delta \right)$$

We rewrite the sums and rearrange the terms in the last line:

$$= \alpha \cdot \beta + \sum_{i=0}^{l} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x) \right)$$
$$+ \sum_{i=l+1}^{m} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x) \right) + h(x)t(x)$$
$$+ \delta \left(s \cdot \left(\sum_{i=0}^{m} a_i u_i(x) + \alpha \right) + r \cdot \left(\sum_{i=0}^{m} a_i v_i(x) + \beta \right) + rs\delta \right)$$

$$= \alpha \cdot \beta + \sum_{i=0}^{l} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x) \right)$$
$$+ \sum_{i=l+1}^{m} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x) \right) + h(x)t(x)$$
$$+ \delta \left(s \cdot \left(\underbrace{\sum_{i=0}^{m} a_i u_i(x) + \alpha + r\delta}_{=A} - r\delta \right) + r \cdot \left(\underbrace{\sum_{i=0}^{m} a_i v_i(x) + \beta + s\delta}_{=B} - s\delta \right) + rs\delta \right)$$

$$= \alpha \cdot \beta + \sum_{i=0}^{l} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x) \right)$$
$$+ \sum_{i=l+1}^{m} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x) \right) + h(x)t(x)$$
$$+ \delta \left(s \cdot (A - r\delta) + r \cdot (B - s\delta) + rs\delta \right)$$

$$
\begin{aligned}
&= \alpha \cdot \beta + \sum_{i=0}^{l} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x)\right) \\
&+ \sum_{i=l+1}^{m} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x)\right) + h(x)t(x) \\
&+ \delta \left(sA - sr\delta + rB - rs\delta + rs\delta\right)
\end{aligned}
$$

$$
= \alpha \cdot \beta + \frac{\sum_{i=0}^{l} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x)\right)}{\gamma} \cdot \gamma
$$

$$
+ \delta \left(\underbrace{\frac{\sum_{i=l+1}^{m} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x)\right) + h(x)t(x)}{\delta} + sA + rB + rs\delta}_{=C} \right)
$$

$$
= \alpha \cdot \beta + \frac{\sum_{i=0}^{l} a_i \left(\alpha v_i(x) + \beta u_i(x) + w_i(x)\right)}{\gamma} \cdot \gamma + \delta \cdot C
$$

□

B. Security Proofs of Kryvos

In this section, we prove the security of Kryvos. In Appendix B.1 we prove verifiability, and in Appendix B.2 tally-hiding of Kryvos.

B.1. Verifiability Proof of Kryvos

In this section, we prove Theorem 5.1 (Verifiability of Kryvos) from Section 5.3.2. The judging procedure of Kryvos is straightforward and works as follows. The judge reads all data from the bulletin board BB and accepts the run if and only if all NIZKPs on BB are valid (and only eligible voters voted, at most once): the voters' NIZKPs $\pi_i^{\mathfrak{C}}$ to prove well-formedness of ballots, the trustees' NIZKP $\pi^{f^{\text{res}}}$ to prove the correctness of the final result, and (if any) the trustees' NIZKP π^{dec} to prove that a voters' ciphertext does not decrypt to a valid opening of her commitments.

Recall that, at a high level, we need to prove the following property. If the final result elecres does not equal $f^{\text{res}}(\vec{H} + \vec{D})$, where $\vec{H}$ consists of all honest voters' choices and $\vec{D}$ consists of at most one (valid) choice per dishonest voter, then the judge J rejects the result.

Essentially, Theorem 5.1 follows from the NIZKPs employed in Kryvos (see Section 5.3.5 for the precise definition of the respective relations).

In what follows, we denote the set of indices of honest voters by I_h and the set of indices by dishonest voters by I_d. Let $(\mathsf{b}_i)_{i \in I_{\mathsf{valid}}}$ be the set of ballots that have not been discarded prior to the homomorphic aggregation at the end of the voting phase.

The first lemma states that honest ballots cannot be discarded prior to the homomorphic aggregation and that, at most, one ballot per dishonest voter exists in the remaining (non-discarded) ballots.

Lemma B.1. *Let I_{valid} be defined as above. Then, we have that $I_h \subseteq I_{\mathsf{valid}}$ and $I_d \supseteq (I_{\mathsf{valid}} \setminus I_h)$ holds (with overwhelming probability in the security parameter η).*

Proof. Observe that $i \in I_{\mathsf{valid}}$ holds for a given ballot b_i if and only if the test executed by the bulletin board BB for incoming ballots (specified at the end of the voting phase, see Section 5.3.1) was successful *and* no trustee T_k published a valid NIZKP proving that e_k^i decrypts to an invalid opening.

Now, to see that $I_h \subseteq I_{\mathsf{valid}}$ holds, observe that the voter's program defined in Section 5.3.1, which each honest voter runs, ensures that the relation of well-formedness $\mathcal{R}_{\mathfrak{C}}$ is satisfied. Therefore, each honest voter's proof $\pi_i^{\mathfrak{C}}$ is valid. Furthermore, due to the IND-CCA2-security

of the PKE encryption scheme (and because the number of voters is polynomially bounded), the probability that an honest voter creates a partially identical ballot to a different voter's is negligible (in the security parameter). This ensures that each honest voter's ballot b_i is appended by the bulletin board (with assumption (V2) in Section 5.3.2 we assume that BB is honest) to the list of ballots $\vec{\mathsf{b}}$. Since the NIZKP of correct decryption π^{dec} is sound and the (honest) voter's program guarantees that each ciphertext e_k^i encrypts a correct opening for the respective commitments, a (possibly dishonest) trustee T_k is not able to create a valid NIZKP for claiming that an honest voter's ciphertext e_k^i decrypts to an invalid opening. Hence, $I_h \subseteq I_{\mathsf{valid}}$ follows (with overwhelming probability).

The fact that $I_d \supseteq (I_{\mathsf{valid}} \setminus I_h)$ holds follows from the checks (precisely, the eligibility and re-voting check) executed by the bulletin board BB before it adds a ballot to $\vec{\mathsf{b}}$. □

The following Lemma states that all ballots not discarded prior to the homomorphic aggregation are well-formed (i.e., *contain* a valid choice from $\mathfrak{C}$) and contain commitments with sufficiently small randomness.

Lemma B.2. *Let I_{valid} be defined as above. Then (with overwhelming probability in the security parameter η), for all $i \in I_{valid}$, there exists $(r_k^{i,l})_{k\in\{1,\ldots,n_{\mathsf{trustees}}\},l\in\{1,\ldots,n_{\mathsf{tuples}}\}}$ and $m^i = (t^{i,j})_{j\in\{1,\ldots,n_{\mathsf{components}}\}} \in \mathfrak{C}$ such that for all $j \in \{1,\ldots,n_{\mathsf{components}}\}$, there exists $(t_k^{i,j})_{k\in\{1,\ldots,n_{\mathsf{trustees}}\}}$ such that*

1. $\forall j \in \{1,\ldots,n_{\mathsf{components}}\}\colon \sum_{k=1}^{n_{\mathsf{trustees}}} t_k^{i,j} \mod q = t^{i,j}$, *and*
2. $\forall k \in \{1,\ldots,n_{\mathsf{trustees}}\}\ \forall l \in \{1,\ldots,n_{\mathsf{tuples}}\}\colon\ c_k^{i,l} = \mathsf{com}((c^{\mathsf{component}}_{(l-1)\cdot N,i,k},\ldots,c^{\mathsf{component}}_{l\cdot N,i,k}), r_k^{i,l})$.

Proof. Recall that the bulletin board (which we assume to be honest) verifies whether each incoming ballot b_i contains valid NIZKP $\pi_i^{\mathfrak{C}}$, and rejects b_i if this is not the case. Therefore, if $i \in I_{\mathsf{valid}}$ for a given b_i, it follows that b_i contains valid ZKP $\pi_i^{\mathfrak{C}}$. Hence, fact (1) and (2) follow from the computational soundness of $\pi_i^{\mathfrak{C}}$. □

The following Lemma states that the homomorphically aggregated commitments correctly open to the total numbers of votes per choice, as determined by I_{valid}.

Lemma B.3. *Let I_{valid} be defined as above. Let $(c_k^{i,l})_{i\in I_{valid},k\in\{1,\ldots,n_{\mathsf{trustees}}\},l\in\{1,\ldots,n_{\mathsf{tuples}}\}}$ be the input commitments to the tallying phase. Then (with overwhelming probability in the security parameter η), for all $l \in \{1,\ldots,n_{\mathsf{tuples}}\}$, there exists $(t_k^{i,l})_{i\in I_{valid},j\in\{1,\ldots,n_{\mathsf{trustees}}\}}$ and $r^{\mathsf{agg},l}$ such that*

$$c^{\mathsf{agg},l} = \mathsf{com}\Big(\sum_{k=1}^{n_{\mathsf{trustees}}} \sum_{i\in I_{valid}} t_k^{i,l}, r^{\mathsf{agg},l}\Big).$$

Proof. This fact follows from Lemma B.2 and the homomorphic property of the commitment scheme. □

Since the trustees' SNARK $\pi^{f^{\text{res}}}$ (correct result) is computationally sound, we can conclude from Lemma B.3 that if the final result does not equal $f^{\text{res}} \circ f^{\mathsf{agg}}((c_i^{\mathsf{choice}})_{i \in I_{\mathsf{valid}}})$, then the judge J rejects the result (with overwhelming probability). Hence, Theorem 5.1 follows from Lemma B.1.

B.2. Tally-Hiding Proof of Kryvos

In this section, we prove Theorem 5.4, which establishes the public and internal privacy levels of Kryvos. We can express these levels using the privacy level of the voting protocol with ideal privacy (see Figure 3.1).

Overview

Recall that, in order to prove the theorem for the protocol Kryvos with n_{voters} voters, n_{trustees} trustees, voting method $(\mathfrak{C}, f^{\text{res}})$, voting distribution μ, and voter under observation $\mathsf{v}_{\mathsf{obs}}$, we have to show the following two conditions:

- *Internal privacy*: For all $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \in \mathfrak{C}$, for all programs $\mathfrak{p}_*$ of the remaining parties such that at least $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters and *at least one* trustee are honest in $\mathfrak{p}_*$ (excluding the voter under observation $\mathsf{v}_{\mathsf{obs}}$), we have that

$$\left|\Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_0^{\mathsf{choice}}) \| \mathfrak{p}_*) \mapsto 1] - \Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_1^{\mathsf{choice}}) \| \mathfrak{p}_*) \mapsto 1]\right|$$

  is $\delta^{\text{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$-bounded as a function of the security parameter η, where f_{AggTally} is the function that returns the aggregated tally (see Section 2.7).

- *Public privacy*: For all $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \in \mathfrak{C}$, for all programs $\mathfrak{p}_*$ of the remaining parties such that at least $n_{\mathsf{voters}}^{\mathsf{honest}}$ voters and *all* trustees are honest in $\mathfrak{p}_*$ (excluding the voter under observation $\mathsf{v}_{\mathsf{obs}}$), we have that

$$\left|\Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_0^{\mathsf{choice}}) \| \mathfrak{p}_*) \mapsto 1] - \Pr[(\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_1^{\mathsf{choice}}) \| \mathfrak{p}_*) \mapsto 1]\right|$$

  is $\delta^{\text{ideal}}_{n_{\mathsf{voters}}, n_{\mathsf{voters}}^{\mathsf{honest}}, \mu}(\mathfrak{C}, f^{\text{res}})$-bounded as a function of the security parameter η, where f^{res} is the actual (tally-hiding) result function.

We can split the composition $\mathfrak{p}_*$ into its honest and its (potentially) dishonest parts. Let HV be the set of all honest voters (without the voter under observation) and $\mathfrak{p}_{\mathsf{HV}}$ be the composition of their honest programs. Recall that the judge J, the scheduler S, the bulletin board BB, and at least one (in case of internal privacy) out of or all (in case of public privacy) n_{trustees} trustees are honest (w.l.o.g., we assume that the first trustee T_1 is honest in both cases). Therefore, we denote the honest part for the internal privacy result by

$$\mathfrak{P}_{\mathsf{honest}} = \mathfrak{p}_{\mathsf{J}} \| \mathfrak{p}_{\mathsf{BB}} \| \mathfrak{p}_{\mathsf{S}} \| \mathfrak{p}_{\mathsf{T}_1} \| \mathfrak{p}_{\mathsf{HV}},$$

and for the public privacy result by

$$\mathfrak{P}_{\mathsf{honest}} = \mathfrak{p}_{\mathsf{J}} \| \mathfrak{p}_{\mathsf{BB}} \| \mathfrak{p}_{\mathsf{S}} \| \mathfrak{p}_{\mathsf{T}_1} \| \ldots \| \mathfrak{p}_{\mathsf{T}_{n_{\mathsf{trustees}}}} \| \mathfrak{p}_{\mathsf{HV}}.$$

By $\mathfrak{P}_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$, we will denote the composition of all honest programs, including the program of the voter under observation $\mathsf{v}_{\mathsf{obs}}$, i.e., $\mathfrak{P}_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) = \mathfrak{P}_{\mathsf{honest}} \| \mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_\star^{\mathsf{choice}})$. All remaining parties are subsumed by the adversarial process $\mathfrak{P}_{\mathsf{A}}$. This means that we can write $\mathfrak{p}_{\mathsf{v}_{\mathsf{obs}}}(c_\star^{\mathsf{choice}}) \| \mathfrak{p}_*$ as $\mathfrak{P}_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) \| \mathfrak{P}_{\mathsf{A}}$.

To prove the result, we use two sequences of games (one for internal and one for external privacy). We fix $c_\star^{\mathsf{choice}} \in \mathfrak{C}$ and start with Game 0, the process $\mathfrak{P}_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}$. Step by step, we transform Game 0 into Game x which is the composition $\mathfrak{P}^x_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}$ for some process $\mathfrak{P}^x_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$ and the same adversarial process $\mathfrak{p}_{\mathsf{A}}$. Game x will be proven indistinguishable from Game 0 from the adversary's point of view, which means that

$$\left| \Pr[(\mathfrak{P}^0_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}^x_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] \right|$$

is negligible for a fixed $c_\star^{\mathsf{choice}} \in \mathfrak{C}$ (as a function of the security parameter).

Furthermore, it will be straightforward to show that in Game x for arbitrary $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \in \mathfrak{C}$, the distance

$$\left| \Pr[(\mathfrak{P}^x_{\mathsf{honest}}(c_0^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}^x_{\mathsf{honest}}(c_1^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] \right|$$

is bounded by $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$ (internal privacy) or $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ (public privacy). The reason is that $\mathfrak{P}^x_{\mathsf{honest}}(c_0^{\mathsf{choice}})$ and $\mathfrak{P}^x_{\mathsf{honest}}(c_1^{\mathsf{choice}})$ use the ideal voting protocol for voting method $(\mathfrak{C}, f_{\mathsf{AggTally}})$ (internal privacy) or $(\mathfrak{C}, f^{\mathrm{res}})$, with $n^{\mathsf{honest}}_{\mathsf{voters}}$ honest voters. Using the triangle inequality, we can therefore deduce that

$$\left| \Pr[(\mathfrak{P}_{\mathsf{honest}}(c_0^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}_{\mathsf{honest}}(c_1^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] \right|$$

is bounded by $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$ (internal privacy) or $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ (public privacy) for all $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \in \mathfrak{C}$ (as a function of the security parameter η).

Notation

We write $\mathfrak{P}^1_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$ for $\mathfrak{P}_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$ and consider $\mathfrak{P}^1_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$ as one atomic process (one program) and not as a composition of processes. We do so is w.l.o.g. since a single program can simulate every (sub-)process. Now, the original Kryvos program is the process $\mathfrak{P}^1_{\mathsf{honest}}(c_\star^{\mathsf{choice}}) \| \mathfrak{p}_{\mathsf{A}}$. For both sequences of games below, we describe how to modify the previous protocol so that any (probabilistic polynomial time) adversary can not distinguish which of these

two protocols he interacts with. For each Game j), we denote the respective honest part of the protocol by $\mathfrak{P}^{j}_{\mathsf{honest}}(c^{\mathsf{choice}}_{\star})$. Let I_h be the set of honest voter indices (excluding the voter under observation).

B.2.1. Internal Privacy

We use a sequence of games to prove Theorem 5.3, i.e., that the internal privacy level of Kryvos is $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$ under the assumptions (P1) and (P2) (see Section 5.3.2) and that at least one trustee is honest (w.l.o.g., we assume that T_1 is honest). The proof aims to employ the ideal voting functionality to compute the honest voters' aggregated choice. We exploit the zero-knowledge property of the ZKPs, the semantic security of the PKE scheme, and the hidingness of the commitment to simulate the honest participants in such a way that the honest participants (in particular the voters) no longer create (and know) the individual honest votes. Instead, the honest participants only use the (aggregated) honest result from the ideal voting functionality. Therefore, even if the adversary can control all honest participants, he only gets to know the aggregated output from the ideal voting functionality. At the same time, we show that all modifications are indistinguishable from the adversary's perspective. This proves that the internal privacy level of Kryvos is negligibly close to the one of the ideal voting functionality, which is $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$. More precisely, our proof works as follows:

Game 0 $\mathfrak{P}^{0}_{\mathsf{honest}}(c^{\mathsf{choice}}_{\star})$ is the original Kryvos protocol.

Game 1 In $\mathfrak{P}^{1}_{\mathsf{honest}}(c^{\mathsf{choice}}_{\star})$, we modify the specification of the (honest) trustee T_1 as follows. If the ciphertext e^{i}_{1} of an arbitrary honest voter v_i decrypts to an invalid opening, then T_1 aborts. The T_1 specification does not change apart from this modification. Due to the correctness of the public-key encryption scheme $\mathcal{E}$, Game 0 and Game 1 are perfectly indistinguishable from the adversary's perspective.

Game 2 In $\mathfrak{P}^{2}_{\mathsf{honest}}(c^{\mathsf{choice}}_{\star})$, we modify the specification of all honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. Each v_i ($i \in I_h$) and $\mathsf{v}_{\mathsf{obs}}$ encrypt (the dummy message) 0^{ϑ} under the honest trustee's, i.e., T_1's, public key pk_1; precisely: $e^{i}_{1} \leftarrow E_{\mathsf{pk}}(\mathsf{pk}_1, 0^{\vartheta})$ (where ϑ is the fixed plaintext size). Apart from this modification, the specification of v_i ($i \in I_h$) and $\mathsf{v}_{\mathsf{obs}}$ does not change. Furthermore, we modify the specification of the honest trustee T_1 as follows. Instead of decrypting the honest voters' ciphertexts e^{i}_{1} (which now encrypt useless dummy messages), the honest trustee T_1 receives the honest voters' opening values from the honest voters internally (inside of the simulator which subsumes all honest parties). Due to the IND-CCA2-security of the public-key encryption scheme $\mathcal{E}$ and the removal of ciphertext duplicates, Game 1 and Game 2 are computationally indistinguishable from the adversary's perspective.

Game 3 In $\mathfrak{P}^{3}_{\mathsf{honest}}(c^{\mathsf{choice}}_{\star})$, we modify the specification of the (honest) voting authority Auth as follows. Recall that the voting authority runs the setup algorithm Setup of the SNARK $\pi^{\mathfrak{C}}_{i}$, which is used to prove ballot validity of the voters' commitments to obtain a pair of common reference string/trapdoor $(\mathsf{CRS}_{\mathfrak{C}}, \tau_{\mathfrak{C}})$. Now, instead of keeping the trapdoor secret,

the authority internally (inside of the simulator) forwards the trapdoor $\tau_{\mathfrak{C}}$ to the voter under observation $\mathsf{v}_{\mathsf{obs}}$ and all honest voters v_i ($i \in I_h$). Game 2 and Game 3 are perfectly indistinguishable from the adversary's perspective.

Game 4 In $\mathfrak{P}^4_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of all honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. Instead of using the common reference string $\mathsf{CRS}_{\mathfrak{C}}$ to run the prover algorithm $\mathsf{Prove}(\mathcal{R}_{\mathfrak{C}}, \mathsf{CRS}_{\mathfrak{C}}, x, w)$, the honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ use the trapdoor $\tau_{\mathfrak{C}}$ (received by Auth in the previous game) to run the simulator algorithm $\mathsf{Sim}(\mathcal{R}_{\mathfrak{C}}, \tau_{\mathfrak{C}}, x)$ of the SNARK $\pi^{\mathfrak{C}}_i$, where (x, w) denotes the respective voter's pair of public input and witness. Apart from these modifications, the specification of v_i ($i \in I_h$) and $\mathsf{v}_{\mathsf{obs}}$ does not change. Game 3 and Game 4 are perfectly indistinguishable from the adversary's perspective due to the perfect zero-knowledge property of the SNARK $\pi^{\mathfrak{C}}_i$.

Game 5 In $\mathfrak{P}^5_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the honest trustee T_1 as follows. The trustee T_1 simulates the NIZKP of correct decryption π^{dec} (if any). Due to the perfect zero-knowledge property of the NIZKP π^{dec}, Game 4 and Game 5 are perfectly indistinguishable.

Game 6 In $\mathfrak{P}^6_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of all honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. The voter under observation $\mathsf{v}_{\mathsf{obs}}$ computes $c^{\mathsf{choice}}_i \xleftarrow{\mu} \mathfrak{C}$ for all honest voters $i \in I_h$, sets $c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{honest}} \leftarrow \sum_{i \in I_h} c^{\mathsf{choice}}_i$, and runs the voting algorithm with "choice" $c^{\mathsf{choice}}_{\mathsf{agg}} \leftarrow c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{honest}} + c^{\mathsf{choice}}_{\mathsf{v}_{\mathsf{obs}}}$ instead of the voter's choice $c^{\mathsf{choice}}_{\mathsf{v}_{\mathsf{obs}}}$. This means that the "choice" contained in $\mathsf{v}_{\mathsf{obs}}$'s ballot is an aggregation of all honest voters' choices plus the one of the voter under observation. At the same time, all honest voters v_i ($i \in I_h$) run the voting algorithm with dummy "choice" 0. Recall that all voters' choices are homomorphically aggregated in the tallying phase. Therefore, it is inconsequential how the honest voters' aggregated choice is split among the homomorphic commitments. Because the commitment scheme is unconditionally hiding, ciphertext duplicates are removed, and full-threshold secret sharing is employed, Game 5 and Game 6 are computationally indistinguishable from the adversary's perspective.

Game 7 In $\mathfrak{P}^7_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. The voter under observation $\mathsf{v}_{\mathsf{obs}}$ invokes the ideal voting functionality $\mathcal{I}_{\mathsf{voting}}$ (see Figure 3.1) to obtain $\mathsf{elecres} \leftarrow \mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f_{\mathsf{AggTally}}, n_{\mathsf{voters}} + 1, n^{\mathsf{honest}}_{\mathsf{voters}} + 1, \mu')$, where the voting distribution μ' always returns $c^{\mathsf{choice}}_{\mathsf{v}_{\mathsf{obs}}}$ for the first (honest) voter and equals μ for the remaining $n^{\mathsf{honest}}_{\mathsf{voters}}$ (honest) voters. Because the ideal voting functionality creates $\mathsf{elecres}$ like $\mathsf{v}_{\mathsf{obs}}$ did in the previous step, Game 6 and Game 7 are perfectly indistinguishable from the adversary's perspective.

In Game 7, we have that for arbitrary $c^{\mathsf{choice}}_0, c^{\mathsf{choice}}_1 \in \mathfrak{C}$, the distance

$$\left|\Pr[(\mathfrak{P}^7_{\mathsf{honest}}(c^{\mathsf{choice}}_0) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}^7_{\mathsf{honest}}(c^{\mathsf{choice}}_1) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1]\right|$$

is bounded by $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$. Since we have proved above that Game 1 (the original Kryvos protocol) and Game 7 are computationally indistinguishable (under the assumptions

made, in particular, that T_1 is honest), we can conclude that

$$\left|\Pr[(\mathfrak{P}_{\mathsf{honest}}(c_0^{\mathsf{choice}})\|\mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}_{\mathsf{honest}}(c_1^{\mathsf{choice}})\|\mathfrak{p}_{\mathsf{A}}) \mapsto 1]\right|$$

is bounded by $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(\mathfrak{C}, f_{\mathsf{AggTally}})$ for all $c_0^{\mathsf{choice}}, c_1^{\mathsf{choice}} \in \mathfrak{C}$ (as a function of the security parameter η).

B.2.2. Public Privacy

We have developed a sequence of games to prove that the external privacy level of Kryvos is $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}},n^{\mathsf{honest}}_{\mathsf{voters}},\mu}(\mathfrak{C}, f^{\mathrm{res}})$. We do so under the assumptions (P1) and (P2) explained in detail in Section 5.3.2. We assume that all trustees are honest, and we assume the existence of a single T_Δ who is honest. We aim to use the ideal voting functionality to compute the (tally-hiding) result function f^{res} based on the dishonest voters' choices.

To achieve this, we exploit the zero-knowledge property of the ZKPs, the semantic security of the PKE scheme, and the hidingness of the commitment. We simulate the honest participants in a way that they no longer create or know the individual honest votes. Instead, they only use the tally-hiding result obtained from the ideal voting functionality. This ensures that even if the adversary can control all honest participants, they only get to know the tally-hiding output from the ideal voting functionality. At the same time, we show that all modifications are indistinguishable from the adversary's perspective.

Our sequence of games works as follows:

Game 0 $\mathfrak{P}^0_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$ is the original Kryvos protocol.

Game 1 In $\mathfrak{P}^1_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$, we modify the specification of the (honest) trustee T_Δ as follows. If the ciphertext e_1^i of an arbitrary honest voter v_i decrypts to an invalid opening, then T_Δ aborts. Apart from this modification, the specification of T_Δ does not change. Due to the correctness of the public-key encryption scheme $\mathcal{E}$, Game 0 and Game 1 are perfectly indistinguishable from the adversary's perspective.

Game 2 In $\mathfrak{P}^2_{\mathsf{honest}}(c_\star^{\mathsf{choice}})$, we modify the specification of all honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. Each v_i ($i \in I_h$) and $\mathsf{v}_{\mathsf{obs}}$ encrypt (the dummy message) 0^ϑ under the trustee's public key pk_1; precisely: $e_1^i \leftarrow E_{\mathsf{pk}}(\mathsf{pk}, 0^\vartheta)$ (where ϑ is the fixed plaintext size). Apart from this modification, the specification of v_i ($i \in I_h$) and $\mathsf{v}_{\mathsf{obs}}$ does not change. Furthermore, we modify the specification of the trustee T_Δ as follows. Instead of decrypting the honest voters' ciphertexts e_1^i (which now encrypt useless dummy messages), the trustee T_Δ receives the honest voters' opening values from the honest voters internally (inside of the simulator which subsumes all honest parties). Due to the IND-CCA2-security of the public-key encryption scheme $\mathcal{E}$ and the removal of ciphertext duplicates, Game 1 and Game 2 are computationally indistinguishable from the adversary's perspective.

Game 3 In $\mathfrak{P}^3_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the honest trustee T_Δ as follows. The trustee T_Δ simulates the NIZKP of correct decryption π^{dec} (if any). Due to the perfect zero-knowledge property of the NIZKP π^{dec}, Game 2 and Game 3 are perfectly indistinguishable.

Game 4 In $\mathfrak{P}^4_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the voting authority Auth. The voting authority runs the setup algorithm Setup of the SNARK $\pi^{f^{\mathrm{res}}}$ to prove the correctness of the final result and obtain a pair of common reference string/trapdoor $(\mathsf{CRS}_{f^{\mathrm{res}}}, \tau_{f^{\mathrm{res}}})$. Previously, the trapdoor τ^{res}_f was kept secret. However, the authority internally forwards the trapdoor τ^{res}_f to the trustee T_Δ inside the simulator. This change does not affect the adversary's perspective, as Game 3 and Game 4 remain perfectly indistinguishable.

Game 5 In $\mathfrak{P}^5_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the trustee T_Δ as follows. Instead of using the common reference string $\mathsf{CRS}_{f^{\mathrm{res}}}$ to run the prover algorithm $\mathsf{Prove}(\mathcal{R}_{f^{\mathrm{res}}}, \mathsf{CRS}_{f^{\mathrm{res}}}, x, w)$, the trustee uses the trapdoor τ^{res}_f (received by Auth in the previous game) to run the simulator algorithms $\mathsf{Sim}(\mathcal{R}_{f^{\mathrm{res}}}, \tau_{f^{\mathrm{res}}}, x)$, where (x, w) denotes the trustee's pair of public input and witness. Apart from these modifications, the specification of T_Δ does not change. Game 4 and Game 5 are perfectly indistinguishable from the adversary's perspective due to the perfect zero-knowledge property of the SNARK $\pi^{f^{\mathrm{res}}}$.

Game 6 In $\mathfrak{P}^6_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the (honest) voting authority Auth as follows. Recall that the voting authority runs the setup algorithm Setup of the SNARK $\pi^{\mathfrak{C}}_i$, which is used to prove ballot validity of the voters' commitments to obtain a pair of common reference string/trapdoor $(\mathsf{CRS}_{\mathfrak{C}}, \tau_{\mathfrak{C}})$. Now, instead of keeping the trapdoor secret, the authority internally (inside of the simulator) forwards the trapdoor $\tau_{\mathfrak{C}}$ to the voter under observation $\mathsf{v}_{\mathsf{obs}}$ and all honest voters v_i $(i \in I_h)$. Game 5 and Game 6 are perfectly indistinguishable from the adversary's perspective.

Game 7 In $\mathfrak{P}^7_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of all honest voters v_i $(i \in I_h)$ and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. Instead of using the common reference string $\mathsf{CRS}_{\mathfrak{C}}$ to run the prover algorithm $\mathsf{Prove}(\mathcal{R}_{\mathfrak{C}}, \mathsf{CRS}_{\mathfrak{C}}, x, w)$, the honest voters v_i $(i \in I_h)$ and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ use the trapdoor $\tau_{\mathfrak{C}}$ (received by Auth in the previous game) to run the simulator algorithm $\mathsf{Sim}(\mathcal{R}_{\mathfrak{C}}, \tau_{\mathfrak{C}}, x)$ of the SNARK$\pi^{\mathfrak{C}}_i$, where (x, w) denotes the respective voter's pair of public input and witness. Apart from these modifications, the specification of v_i $(i \in I_h)$ and $\mathsf{v}_{\mathsf{obs}}$ does not change. Game 6 and Game 7 are perfectly indistinguishable from the adversary's perspective due to the perfect zero-knowledge property of the SNARK $\pi^{\mathfrak{C}}_i$.

Game 8 In $\mathfrak{P}^8_{\mathsf{honest}}(c^{\mathsf{choice}}_\star)$, we modify the specification of the honest trustee T_Δ as follows. The trustee T_Δ sets $c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{dishonest}}$ to be the (aggregated) dishonest choices (observe that T_Δ can decrypt/open all dishonest voters' valid choices). For all $i \in I_h$, the trustee computes $c^{\mathsf{choice}}_{\mathsf{agg}}(i) \xleftarrow{\mu} \mathfrak{C}$. Then, T_Δ sets $c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{honest}} \leftarrow \sum_{i \in I_h} c^{\mathsf{choice}}_{\mathsf{agg}}(i)$ and $c^{\mathsf{choice}}_{\mathsf{agg}} \leftarrow c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{honest}} + c^{\mathsf{choice}}_{\mathsf{v}_{\mathsf{obs}}}$, and returns the final result $\mathsf{elecres} \leftarrow f^{\mathrm{res}}(c^{\mathsf{choice}}_{\mathsf{agg}} \cup c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{dishonest}})$. Because the commitment scheme is unconditionally hiding, Game 7 and Game 8 are perfectly indistinguishable from the adversary's perspective.

Game 9 In $\mathfrak{P}^{9}_{\mathsf{honest}}(c^{\mathsf{choice}}_{\star})$, we modify the specification of all honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ as follows. All honest voters v_i ($i \in I_h$) and the voter under observation $\mathsf{v}_{\mathsf{obs}}$ run the voting algorithm with dummy "choice" 0. Because the commitment scheme is unconditionally hiding, Game 8 and Game 9 are perfectly indistinguishable from the adversary's perspective.

Game 10 In $\mathfrak{P}^{1}_{\mathsf{honest}}0(c^{\mathsf{choice}}_{\star})$, we modify the specification of the trustee T_{Δ} as follows. Trustee T_{Δ} invokes the ideal voting functionality $\mathcal{I}_{\mathsf{voting}}$ to obtain $\mathsf{elecres} \leftarrow \mathcal{I}_{\mathsf{voting}}(\mathfrak{C}, f^{\mathrm{res}}, n_{\mathsf{voters}} + 1, n^{\mathsf{honest}}_{\mathsf{voters}} + 1, \mu')$, where the voting distribution μ' always returns $c^{\mathsf{choice}}_{\mathsf{v}_{\mathsf{obs}}}$ for the first (honest) voter and equals μ for the remaining $n^{\mathsf{honest}}_{\mathsf{voters}}$ (honest) voters, and where the dishonest choices are extracted from $c^{\mathsf{choice}}_{\mathsf{agg},\mathsf{dishonest}}$. Because the ideal voting functionality creates $\mathsf{elecres}$ like T_{Δ} did in the previous step, Game 9 and Game 10 are perfectly indistinguishable from the adversary's perspective.

In Game 10, we have that for arbitrary $c^{\mathsf{choice}}_{0}, c^{\mathsf{choice}}_{1} \in \mathfrak{C}$, the distance

$$\left|\Pr[(\mathfrak{P}^{10}_{\mathsf{honest}}(c^{\mathsf{choice}}_{0}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}^{10}_{\mathsf{honest}}(c^{\mathsf{choice}}_{1}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1]\right|$$

is bounded by $\delta^{\mathrm{ideal}}_{(n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu)}(\mathfrak{C}, f^{\mathrm{res}})$. Since we have proved above that Game 1 (the original Kryvos protocol) and Game 10 are indistinguishable (under the assumptions made, in particular, that all trustees are honest), we can conclude that

$$\left|\Pr[(\mathfrak{P}_{\mathsf{honest}}(c^{\mathsf{choice}}_{0}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1] - \Pr[(\mathfrak{P}_{\mathsf{honest}}(c^{\mathsf{choice}}_{1}) \| \mathfrak{p}_{\mathsf{A}}) \mapsto 1]\right|$$

is information-theoretically bounded by $\delta^{\mathrm{ideal}}_{n_{\mathsf{voters}}, n^{\mathsf{honest}}_{\mathsf{voters}}, \mu}(\mathfrak{C}, f^{\mathrm{res}})$ for all $c^{\mathsf{choice}}_{0}, c^{\mathsf{choice}}_{1} \in \mathfrak{C}$.

Bibliography

[ABL+19] Behzad Abdolmaleki, Karim Baghery, Helger Lipmaa, Janno Siim, and Michal Zajac. UC-Secure CRS Generation for SNARKs. In *Progress in Cryptology - AFRICACRYPT 2019 - 11th International Conference on Cryptology in Africa, Rabat, Morocco, July 9-11, 2019, Proceedings*, volume 11627 of *Lecture Notes in Computer Science*, pages 99–117. Springer, 2019.

[ABST23] Miguel Ambrona, Marc Beunardeau, Anne-Laure Schmitt, and Raphael R. Toledo. a$PlonK$: Aggregated $PlonK$ from Multi-polynomial Commitment Schemes. In *Advances in Information and Computer Security - 18th International Workshop on Security, IWSEC 2023, Yokohama, Japan, August 29-31, 2023, Proceedings*, volume 14128 of *Lecture Notes in Computer Science*, pages 195–213. Springer, 2023.

[AC20] Thomas Attema and Ronald Cramer. Compressed Σ-Protocol Theory and Practical Application to Plug & Play Secure Algorithmics. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 513–543. Springer, 2020.

[ACC+22] Thomas Attema, Ignacio Cascudo, Ronald Cramer, Ivan Damgård, and Daniel Escudero. Vector Commitments over Rings and Compressed Σ-Protocols. In *Theory of Cryptography - 20th International Conference, TCC 2022, Chicago, IL, USA, November 7-10, 2022, Proceedings, Part I*, volume 13747 of *Lecture Notes in Computer Science*, pages 173–202. Springer, 2022.

[ACK21] Thomas Attema, Ronald Cramer, and Lisa Kohl. A Compressed Σ-Protocol Theory for Lattices. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 549–579. Springer, 2021.

[ACL+22] Martin R. Albrecht, Valerio Cini, Russell W. F. Lai, Giulio Malavolta, and Sri Aravinda Krishnan Thyagarajan. Lattice-Based SNARKs: Publicly Verifiable, Preprocessing, and Recursively Composable - (Extended Abstract). In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part*

II, volume 13508 of *Lecture Notes in Computer Science*, pages 102–132. Springer, 2022.

[ACM20] ACM. Q&A on ACM's Internet Voting. `https://www.acm.org/binaries/content/assets/acmelections/acminternetvoting-1.pdf` (accessed 19.06.2023), 2020.

[Adi08] Ben Adida. Helios: Web-based Open-Audit Voting. In Paul C. van Oorschot, editor, *Proceedings of the 17th USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[AdMPQ09] Ben Adida, Olivier de Marneffe, Olivier Pereira, and Jean-Jaques Quisquater. Electing a University President Using Open-Audit Voting: Analysis of Real-World Use of Helios. In *USENIX/ACCURATE Electronic Voting Technology (EVT 2009)*, 2009.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 2087–2104. ACM, 2017.

[AL21] Martin R. Albrecht and Russell W. F. Lai. Subtractive Sets over Cyclotomic Rings - Limits of Schnorr-Like Arguments over Lattices. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II*, volume 12826 of *Lecture Notes in Computer Science*, pages 519–548. Springer, 2021.

[ALS20] Thomas Attema, Vadim Lyubashevsky, and Gregor Seiler. Practical Product Proofs for Lattice Commitments. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 470–499. Springer, 2020.

[ALSZ21] Behzad Abdolmaleki, Helger Lipmaa, Janno Siim, and Michal Zajac. On Subversion-Resistant SNARKs. *J. Cryptol.*, 34(3):17, 2021.

[AMD20] AMD. AMD Secure Encrypted Virtualization. `https://developer.amd.com/sev/` (accessed 19.06.2023), 2020.

[ASI06] AK Agarwala, DT Shahani, and PV Indiresan. Report of the expert committee for evaluation of the upgraded electronic voting machine (EVM), 2006.

[BB89] Judit Bar-Ilan and Donald Beaver. Non-Cryptographic Fault-Tolerant Computing in Constant Number of Rounds of Interaction. In *Proceedings of the Eighth Annual*

ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989, pages 201–209. ACM, 1989.

[BBB+13] Susan Bell, Josh Benaloh, Mike Byrne, Dana DeBeauvoir, Bryce Eakin, Gail Fischer, Philip Kortum, Neal McBurnett, Julian Montoya, Michelle Parker, Olivier Pereira, Philip Stark, Dan Wallach, , and Michael Winn. STAR-Vote: A Secure, Transparent, Auditable, and Reliable Voting System. *USENIX Journal of Election Technology and Systems (JETS)*, 1:18–37, August 2013.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short Proofs for Confidential Transactions and More. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334. IEEE Computer Society, 2018.

[BBC+18] Carsten Baum, Jonathan Bootle, Andrea Cerulli, Rafaël del Pino, Jens Groth, and Vadim Lyubashevsky. Sub-linear Lattice-Based Zero-Knowledge Arguments for Arithmetic Circuits. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part II*, volume 10992 of *Lecture Notes in Computer Science*, pages 669–699. Springer, 2018.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Technical Report 2018/46, Cryptology ePrint Archive, 2018.

[BCG+15a] Eli Ben-Sasson, Alessandro Chiesa, Matthew Green, Eran Tromer, and Madars Virza. Secure Sampling of Public Parameters for Succinct Zero Knowledge Proofs. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 287–304. IEEE Computer Society, 2015.

[BCG+15b] David Bernhard, Véronique Cortier, David Galindo, Olivier Pereira, and Bogdan Warinschi. SoK: A Comprehensive Analysis of Game-Based Ballot Privacy Definitions. In *2015 IEEE Symposium on Security and Privacy, SP 2015, San Jose, CA, USA, May 17-21, 2015*, pages 499–516. IEEE Computer Society, 2015.

[BCH+12] Craig Burton, Chris Culnane, James Heather, Thea Peacock, Peter Y. A. Ryan, Steve A. Schneider, Vanessa Teague, Roland Wen, Zhe Xia, and Sriramkrishnan Srinivasan. Using Prêt à Voter in Victoria State Elections. In *2012 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '12, Bellevue, WA, USA, August 6-7, 2012*. USENIX Association, 2012.

[BCR+19] Eli Ben-Sasson, Alessandro Chiesa, Michael Riabzev, Nicholas Spooner, Madars Virza, and Nicholas P. Ward. Aurora: Transparent Succinct Arguments for R1CS.

In *Advances in Cryptology - EUROCRYPT 2019 - 38th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Darmstadt, Germany, May 19-23, 2019, Proceedings, Part I*, volume 11476 of *Lecture Notes in Computer Science*, pages 103–128. Springer, 2019.

[BCS21] Jonathan Bootle, Alessandro Chiesa, and Katerina Sotiraki. Sumcheck Arguments and Their Applications. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part I*, volume 12825 of *Lecture Notes in Computer Science*, pages 742–773. Springer, 2021.

[Ben86] J. D. Benaloh. Improving Privacy in Cryptographic Elections (technical report). Technical report, Technical report, 1986.

[Ben87] J.G. Benaloh. *Verifiable Secret Ballot Elections*. PhD thesis, Yale University, 1987.

[Ben07] Josh Benaloh. Ballot Casting Assurance via Voter-Initiated Poll Station Auditing. In *2007 USENIX/ACCURATE Electronic Voting Technology Workshop, EVT'07, Boston, MA, USA, August 6, 2007*. USENIX Association, 2007.

[Ber06] Daniel J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Public Key Cryptography - PKC 2006, 9th International Conference on Theory and Practice of Public-Key Cryptography, New York, NY, USA, April 24-26, 2006, Proceedings*, volume 3958 of *Lecture Notes in Computer Science*, pages 207–228. Springer, 2006.

[BFH+20] Rishabh Bhadauria, Zhiyong Fang, Carmit Hazay, Muthuramakrishnan Venkitasubramaniam, Tiancheng Xie, and Yupeng Zhang. Ligero++: A New Optimized Sublinear IOP. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 2025–2038. ACM, 2020.

[BFS16] Mihir Bellare, Georg Fuchsbauer, and Alessandra Scafuro. NIZKs with an Untrusted CRS: Security in the Face of Parameter Subversion. In *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 777–804, 2016.

[BG13] Nils-Christian Bormann and Matt Golder. Democratic electoral systems around the world, 1946–2011. *Electoral Studies*, 32(2):360–369, 2013.

[BG18] Jonathan Bootle and Jens Groth. Efficient Batch Zero-Knowledge Arguments for Low Degree Polynomials. In *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio*

de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II, volume 10770 of *Lecture Notes in Computer Science*, pages 561–588. Springer, 2018.

[BGM17] Sean Bowe, Ariel Gabizon, and Ian Miers. Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model. Technical Report 2017/1050, Cryptology ePrint Archive, 2017.

[BL07] Michel Balinski and Rida Laraki. A Theory of Measuring, Electing, and Ranking. *Proceedings of the National Academy of Sciences*, 104(21):8720–8725, 2007.

[BL10] Michel Balinski and Rida Laraki. Election by majority judgment: experimental evidence. In *In situ and laboratory experiments on electoral law reform: French presidential elections*, pages 13–54. Springer, 2010.

[BL14] Michel Balinski and Rida Laraki. Judge: Don't vote! *Operations Research*, 62:483–511, 06 2014.

[BLNS20] Jonathan Bootle, Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Gregor Seiler. A Non-PCP Approach to Succinct Quantum-Safe Zero-Knowledge. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part II*, volume 12171 of *Lecture Notes in Computer Science*, pages 441–469. Springer, 2020.

[BMN+09] Josh Benaloh, Tal Moran, Lee Naish, Kim Ramchen, and Vanessa Teague. Shuffle-sum: coercion-resistant verifiable tallying for STV voting. *IEEE Trans. Inf. Forensics Secur.*, 4(4):685–698, 2009.

[Bow07] Debra Bowen. Top-to-bottom review of electronic voting systems certified for use in california elections. `https://votingsystems.cdn.sos.ca.gov/oversight/ttbr/draft-top-to-bottom-review.pdf` (accessed 19.06.2023), 2007.

[BS23] Ward Beullens and Gregor Seiler. LaBRADOR: Compact Proofs for R1CS from Module-SIS. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V*, volume 14085 of *Lecture Notes in Computer Science*, pages 518–548. Springer, 2023.

[BSI21] BSI. echnische Richtlinie TR-03162,, 2021. `https://www.bsi.bund.de/DE/Themen/Unternehmen-und-Organisationen/Standards-und-Zertifizierung/Technische-Richtlinien/TR-nach-Thema-sortiert/tr03162/TR-03162_node.html` (accessed 19.06.2023).

[BTP06] Benjamin Beckmann, Götz Trenkler, and Friedrich Pukelsheim. *Das Landtagswahlsystem in Nordrhein-Westfalen.* PhD thesis, Diploma thesis, Universität Dortmund, 2006.

[Bun09] Bundesverfassungsgericht. Judgment of 3 march 2009 - 2 bvc 3/07. `https://www.bundesverfassungsgericht.de/SharedDocs/Entscheidungen/EN/2009/03/cs20090303_2bvc000307en.html` (accessed 19.06.2023), 2009.

[Bun20] Bundesrepublik Deutschland. Bundeswahlgesetz. `https://www.bundeswahlleiter.de/dam/jcr/2596ba8d-34e4-4c9b-a731-a27f8fb0618f/bundeswahlgesetz.pdf` (accessed 19.06.2023), 2020.

[Bun23] Bundesministerium der Justiz. Bundeswahlgesetz: § 6 Wahl nach Landeslisten, 2023. `https://www.gesetze-im-internet.de/bwahlg/__6.html`.

[Bur19] Tom Burt, 2019. `https://blogs.microsoft.com/on-the-issues/2019/09/24/electionguard-available-today-to-enable-secure-verifiable-voting/` (accessed 19.06.2023).

[Can01] Ran Canetti. Universally Composable Security: A New Paradigm for Cryptographic Protocols. In *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS 2001)*, pages 136–145. IEEE Computer Society, 2001.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates. In *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part II*, volume 14005 of *Lecture Notes in Computer Science*, pages 499–530. Springer, 2023.

[CCC+08] D. Chaum, R. Carback, J. Clark, A. Essex, S. Popoveniuc, R. L. Rivest, P. Y. A. Ryan, E. Shen, and A. T. Sherman. Scantegrity II: End-to-End Verifiability for Optical Scan Election Systems using Invisible Ink Confirmation Codes. In *USENIX/ACCURATE Electronic Voting Technology (EVT 2008)*. USENIX Association, 2008. See also `http://www.scantegrity.org/elections.php`.

[CCM08] M. R. Clarkson, S. Chong, and A. C. Myers. Civitas: Toward a Secure Voting System. In *2008 IEEE Symposium on Security and Privacy (S&P 2008)*, pages 354–368. IEEE Computer Society, 2008.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. Proofs of Partial Knowledge and Simplified Design of Witness Hiding Protocols. In *Advances in Cryptology - CRYPTO '94, Proceedings*, pages 174–187, 1994.

[CFL83a] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Lower Bounds for Constant Depth Circuits for Prefix Problems. In *Automata, Languages and Programming, 10th Colloquium, Barcelona, Spain, July 18-22, 1983, Proceedings*, volume 154 of *Lecture Notes in Computer Science*, pages 109–117. Springer, 1983.

[CFL83b] Ashok K. Chandra, Steven Fortune, and Richard J. Lipton. Unbounded Fan-in Circuits and Associative Functions. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing, 25-27 April, 1983, Boston, Massachusetts, USA*, pages 52–60. ACM, 1983.

[CFQ19] Matteo Campanelli, Dario Fiore, and Anaïs Querol. LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2075–2092. ACM, 2019.

[CFSY96] Ronald Cramer, Matthew K. Franklin, Berry Schoenmakers, and Moti Yung. Multi-Authority Secret-Ballot Elections with Linear Work. In *EUROCRYPT 1996*, pages 72–83, 1996.

[CGG+23] Arka Rai Choudhuri, Sanjam Garg, Aarushi Goel, Sruthi Sekar, and Rohit Sinha. SublonK: Sublinear Prover PlonK. Technical Report 2023/902, Cryptology ePrint Archive, 2023.

[CGGI14] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Election Verifiability for Helios under Weaker Trust Assumptions. In *Computer Security - ESORICS 2014 - 19th European Symposium on Research in Computer Security, Wroclaw, Poland, September 7-11, 2014. Proceedings, Part II*, volume 8713 of *Lecture Notes in Computer Science*, pages 327–344. Springer, 2014.

[CGI+23] Shumo Chu, Brandon H. Gomes, Francisco Hernandez Iglesias, Todd Norton, and Duncan Tebbs. UniPlonk: Plonk with Universal Verifier. Technical Report 2023/869, Cryptology ePrint Archive, 2023.

[CGK+16] Véronique Cortier, David Galindo, Ralf Küsters, Johannes Müller, and Tomasz Truderung. SoK: Verifiability Notions for E-Voting Protocols. In *IEEE Symposium on Security and Privacy, SP 2016, San Jose, CA, USA, May 22-26, 2016*, pages 779–798. IEEE Computer Society, 2016.

[CGY22] Véronique Cortier, Pierrick Gaudry, and Quentin Yang. A Toolbox for Verifiable Tally-Hiding E-Voting Systems. In *Computer Security - ESORICS 2022 - 27th European Symposium on Research in Computer Security, Copenhagen, Denmark, September 26-30, 2022, Proceedings, Part II*, volume 13555 of *Lecture Notes in Computer Science*, pages 631–652. Springer, 2022.

[CHJ+22] HeeWon Chung, Kyoohyung Han, Chanyang Ju, Myungsun Kim, and Jae Hong Seo. Bulletproofs+: Shorter Proofs for a Privacy-Enhanced Distributed Ledger. *IEEE Access*, 10:42067–42082, 2022.

[Civ23] Civica. Election Management Services. `https://www.civica.com/en-gb/civica-election-services/` (accessed 19.06.2023), 2023.

[CM05] Michael R. Clarkson and Andrew C. Myers. Coercion-Resistant Remote Voting Using Decryption Mixes. In *In Frontiers in Electronic Elections (FEE 2005)*, 2005.

[Com07] Commonwealth of Pennsylvania. Re-Examnination results of the advanced voting solutions WINvote direct recording electronic voting system with WINware election management software, version 2.0.03. `https://verifiedvoting.org/wp-content/uploads/2020/08/PA_advanced_voting_solutions_winvote_reexam.pdf` (accessed 19.06.2023), 2007.

[COS20] Alessandro Chiesa, Dev Ojha, and Nicholas Spooner. Fractal: Post-quantum and Transparent Recursive Proofs from Holography. In *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 769–793. Springer, 2020.

[Cou13] Council of Europe. Council of Europe adopts new Recommendation on Standards for E-Voting, 2013. `https://www.coe.int/en/web/electoral-assistance/-/council-of-europe-adopts-new-recommendation-on-standards-for-e-voting` (accessed 19.06.2023).

[CPST18] Sébastien Canard, David Pointcheval, Quentin Santos, and Jacques Traoré. Practical Strategy-Resistant Privacy-Preserving Elections. In *ESORICS 2018*, pages 331–349, 2018.

[Cro19] CrossRef. Election process and results. `https://www.crossref.org/board-and-governance/elections/` (accessed 19.06.2023), 2019.

[CRS05] D. Chaum, P.Y.A. Ryan, and S. Schneider. A Practical, Voter-verifiable Election Scheme. In *Proceedings of the 10th European Symposium on Research in Computer Security (ESORICS 2005)*, volume 3679 of *Lecture Notes in Computer Science*, pages 118–139. Springer, 2005.

[CRST15] Chris Culnane, Peter Y. A. Ryan, Steve A. Schneider, and Vanessa Teague. vVote: A Verifiable Voting System. *ACM Trans. Inf. Syst. Secur.*, 18(1):3:1–3:30, 2015.

[CS03] Jan Camenisch and Victor Shoup. Practical Verifiable Encryption and Decryption of Discrete Logarithms. In *Advances in Cryptology - CRYPTO 2003, 23rd Annual*

International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings, volume 2729 of *Lecture Notes in Computer Science*, pages 126–144. Springer, 2003.

[CS11] Véronique Cortier and Ben Smyth. Attacking and Fixing Helios: An Analysis of Ballot Secrecy. In *Proceedings of the 24th IEEE Computer Security Foundations Symposium, CSF, 2011*, pages 297–311, 2011.

[CS14] Chris Culnane and Steve A. Schneider. A Peered Bulletin Board for Robust Use in Verifiable Voting Systems. In *IEEE CSF 2014*, pages 169–183, 2014.

[CS18] Craig Costello and Benjamin Smith. Montgomery curves and their arithmetic - The case of large characteristic fields. *J. Cryptogr. Eng.*, 8(3):227–240, 2018.

[ctV23] ccess the Vote. Top-to-bottom review of electronic voting systems certified for use in california elections. `https://www.accessthevote.org/election-issue/using-paper-ballots/` (accessed 19.06.2023), 2023.

[Deb12] Debian Project. Ubuntu IRC Council Position. `https://www.debian.org/vote/` (accessed 19.06.2023), 2012.

[deb22] debian, 2022. `https://www.debian.org/vote/` (accessed 19.06.2023).

[Der21] Der Bundeswahlleiter. Wahl zum 20. Deutschen Bundestag am 26. September 2021: Heft 3 Endgültige Ergebnisse nach Wahlkreisen, 2021. `https://bundeswahlleiter.de/dam/jcr/cbceef6c-19ec-437b-a894-3611be8ae886/btw21_heft3.pdf`, `https://www.bundeswahlleiter.de/bundestagswahlen/2021/ergebnisse/opendata/csv/` (accessed 19.06.2023).

[Deu] Deutscher Bundestag. Hare-Niemeyer-Verfahren, Sitzverteilung. `https://www.bundestag.de/services/glossar/glossar/A/auszaehlverfahren-hare-niemeyer-854904` (accessed 19.06.2023).

[Deu19] Deutsche Forschungsgemeinschaft (DFG). DFG Fachkollegienwahl 2019. `https://www.dfg.de/download/pdf/dfg_im_profil/gremien/fachkollegien/fk-wahl2019/fkwahl_2019_wahlergebnis_endgueltig_200218.pdf` (accessed 19.06.2023), 2019.

[DFK+06] Ivan Damgård, Matthias Fitzi, Eike Kiltz, Jesper Buus Nielsen, and Tomas Toft. Unconditionally Secure Constant-Rounds Multi-party Computation for Equality, Comparison, Bits and Exponentiation. In *Theory of Cryptography, Third Theory of Cryptography Conference, TCC 2006, New York, NY, USA, March 4-7, 2006, Proceedings*, volume 3876 of *Lecture Notes in Computer Science*, pages 285–304. Springer, 2006.

[DJN10] Ivan Damgård, Mads Jurik, and Jesper Buus Nielsen. A generalization of Paillier's public-key system with applications to electronic voting. *Int. J. Inf. Sec.*, 9(6):371–385, 2010.

[DM04] Wolfgang Drechsler and Ülle Madise. Electronic voting in estonia. In *Electronic voting and democracy: A comparative analysis*, pages 97–108. Springer, 2004.

[Eag22] Liam Eagen. Bulletproofs++. Technical Report 2022/510, Cryptology ePrint Archive, 2022.

[Eco22] The Economist. A wild gerrymander makes hungary's fidesz party hard to dislodge. `https://www.economist.com/graphic-detail/2022/04/02/a-wild-gerrymander-makes-hungarys-fidesz-party-hard-to-dislodge` (accessed 19.06.2023), 2022.

[Ele15] Electoral Commission NSW. NSW State Electoin Results 2015. `https://pastvtr.elections.nsw.gov.au/SGE2015/la-home.htm` (accessed 19.06.2023), 2015.

[Ely12] Elysee. The Constitution of the Fifth Republic. `https://www.elysee.fr/en/french-presidency/constitution-of-4-october-1958` (accessed 19.06.2023), 2012.

[ENS20] Muhammed F. Esgin, Ngoc Khanh Nguyen, and Gregor Seiler. Practical Exact Proofs from Lattices: New Techniques to Exploit Fully-Splitting Rings. In *Advances in Cryptology - ASIACRYPT 2020 - 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7-11, 2020, Proceedings, Part II*, volume 12492 of *Lecture Notes in Computer Science*, pages 259–288. Springer, 2020.

[Eps15] Jeremy Epstein. Weakness in Depth: A Voting Machine's Demise. *IEEE Secur. Priv.*, 13(3):55–58, 2015.

[ESLL19] Muhammed F. Esgin, Ron Steinfeld, Joseph K. Liu, and Dongxi Liu. Lattice-Based Zero-Knowledge Proofs: New Techniques for Shorter and Faster Constructions and Applications. In *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 115–146. Springer, 2019.

[ESS+19] Muhammed F. Esgin, Ron Steinfeld, Amin Sakzad, Joseph K. Liu, and Dongxi Liu. Short Lattice-Based One-out-of-Many Proofs and Applications to Ring Signatures. In *Applied Cryptography and Network Security - 17th International Conference, ACNS 2019, Bogota, Colombia, June 5-7, 2019, Proceedings*, volume 11464 of *Lecture Notes in Computer Science*, pages 67–88. Springer, 2019.

[Eur20] European Broadcasting Union. Eurovision Song Contest - How it works. https://eurovision.tv/about/how-it-works (accessed 19.06.2023), 2020.

[fCR10] International Association for Cryptologic Research, 2010. https://www.iacr.org/elections/eVoting/ (accessed 19.06.2023).

[fDI14] International Institute for Democracy and Electoral Assistance (International IDEA). Proposed basic law on elections and referendums - tunisia (non-official translation to english). https://aceproject.org/ero-en/regions/africa/TN/tunisia-organic-law-on-elections-and-referenda/ (accessed 19.06.2023), 2014.

[fDI23] International Institute for Democracy and Electoral Assistance (International IDEA). Electoral system for national legislature. https://www.idea.int/answer/ans9380463083896 (accessed 19.06.2023), 2023.

[FHR21] Prastudy Fauzi, Martha Norberg Hovd, and Håvard Raddum. A Practical Adaptive Key Recovery Attack on the LGM (GSW-like) Cryptosystem. In *Post-Quantum Cryptography - 12th International Workshop, PQCrypto 2021, Daejeon, South Korea, July 20-22, 2021, Proceedings*, volume 12841 of *Lecture Notes in Computer Science*, pages 483–498. Springer, 2021.

[For14] O. Forster. *Algorithmic Number Theory*. Vieweg+Teubner Verlag, 2014.

[FQZ+21] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: Efficient Zero-Knowledge Proofs for Neural Networks. Technical Report 2021/87, Cryptology ePrint Archive, 2021.

[Gal92] Michael Gallagher. Comparing proportional representation electoral systems: Quotas, thresholds, paradoxes and majorities. *British Journal of Political Science*, 22(4):469–496, 1992.

[Gen84] Georg Genssler. *Das d'Hondtsche Verfahren und andere Sitzverteilungsverfahren aus mathematischer und verfassungsrechtlicher Sicht*. PhD thesis, Universität Regensburg, 1984.

[Ges19] Gesellschaft für Informatik (GI). GI Wahlen. https://gi.de/wahlen (accessed 19.06.2023), 2019.

[GGP15] David Galindo, Sandra Guasch, and Jordi Puiggali. 2015 Neuchâtel's Cast-as-Intended Verification Mechanism. In *VoteID 2015, Proceedings*, pages 3–18, 2015.

[Gjø11] Kristian Gjøsteen. The Norwegian Internet Voting Protocol. In *E-Voting and Identity - Third International Conference, VoteID 2011, Tallinn, Estonia, September 28-30, 2011, Revised Selected Papers*, volume 7187 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2011.

[GLS+23] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-Time and Field-Agnostic SNARKs for R1CS. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part II*, volume 14082 of *Lecture Notes in Computer Science*, pages 193–226. Springer, 2023.

[GMO16] Irene Giacomelli, Jesper Madsen, and Claudio Orlandi. ZKBoo: Faster Zero-Knowledge for Boolean Circuits. In *25th USENIX Security Symposium, 2016*, pages 1069–1083. USENIX Association, 2016.

[Goo14] Nicole Goodman. Internet voting in a local election in Canada. In *The internet and democracy in global perspective: Voters, candidates, parties, and social movements*, pages 7–24. Springer, 2014.

[Gov20] Government of India. Constitution of India. `https://www.india.gov.in/my-government/constitution-india` (accessed 19.06.2023), 2020.

[Gro16] Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In *Advances in Cryptology - EUROCRYPT 2016 - 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge. Technical Report 2019/953, Cryptology ePrint Archive, 2019.

[Hae08] Andreas Udo De Haes. E-voting banned by dutch government. `https://www.itworldcanada.com/article/e-voting-banned-by-dutch-government/346` (accessed 19.06.2023), 2008.

[HAM22] Hina Binte Haq, Syed Taha Ali, and Ronan McDermott. End-to-end verifiable voting for developing countries - what's hard in Lausanne is harder still in Lahore. *CoRR*, abs/2210.04764, 2022.

[Hea07] James Heather. Implementing STV securely in Prêt à Voter. In *IEEE CSF 2007*, pages 157–169, 2007.

[Hei17] Rebecca Heilweil, 2017. `https://www.forbes.com/sites/rebeccaheilweil1/2017/12/02/eight-companies-that-want-to-revolutionize-voting-technology/#29da2b3712c1` (accessed 19.06.2023).

[HGM+22] Helen A. Hayes, Nicole Goodman, R. Michael McGregor, Zachary Spicer, and Scott Pruysers. The Effect of Exogenous Shocks on the Administration of Online Voting:

Evidence from Ontario, Canada. In *Electronic Voting - 7th International Joint Conference, E-Vote-ID 2022, Bregenz, Austria, October 4-7, 2022, Proceedings*, volume 13553 of *Lecture Notes in Computer Science*, pages 70–89. Springer, 2022.

[HHK+21a] Fabian Hertel, Nicolas Huber, Jonas Kittelberger, Ralf Küsters, Julian Liedtke, and Daniel Rausch. Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting. In *Proceedings of E-Vote-ID 2021*, page 269–284. University of Tartu Press, 2021.

[HHK+21b] Fabian Hertel, Nicolas Huber, Jonas Kittelberger, Ralf Küsters, Julian Liedtke, and Daniel Rausch. Extending the Tally-Hiding Ordinos System: Implementations for Borda, Hare-Niemeyer, Condorcet, and Instant-Runoff Voting. Technical Report 2021/1420, Cryptology ePrint Archive, 2021.

[HK02] Alejandro Hevia and Marcos A. Kiwi. Electronic Jury Voting Protocols. In *LATIN 2002, Proceedings*, pages 415–429, 2002.

[HK04] Alejandro Hevia and Marcos A. Kiwi. Electronic jury voting protocols. *Theor. Comput. Sci.*, 321(1):73–94, 2004.

[HKK+22a] Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, CCS 2022, Los Angeles, CA, USA, November 7-11, 2022*, pages 1443–1457. ACM, 2022.

[HKK+22b] Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. Technical Report 2022/1132, Cryptology ePrint Archive, 2022.

[HKK+23] Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. Implementation of Kryvos., 2023. `https://github.com/JulianLiedtke/Kryvos-PhD`.

[HLPT20] Thomas Haines, Sarah Jamie Lewis, Olivier Pereira, and Vanessa Teague. How not to prove your election outcome. In *2020 IEEE Symposium on Security and Privacy, SP 2020, San Francisco, CA, USA, May 18-21, 2020*, pages 644–660. IEEE, 2020.

[HMR+19] Carmit Hazay, Gert Læssøe Mikkelsen, Tal Rabin, Tomas Toft, and Angelo Agatino Nicolosi. Efficient RSA Key Generation and Threshold Paillier in the Two-Party Setting. *J. Cryptol.*, 32(2):265–323, 2019.

[HMVW16] Sven Heiberg, Tarvi Martens, Priit Vinkel, and Jan Willemson. Improving the Verifiability of the Estonian Internet Voting Scheme. In *Electronic Voting - First*

International Joint Conference, E-Vote-ID 2016, Bregenz, Austria, October 18-21, 2016, Proceedings, volume 10141 of *Lecture Notes in Computer Science*, pages 92–107. Springer, 2016.

[Hou10] Stefan Hougardy. The Floyd-Warshall algorithm on graphs with negative cycles. *Inf. Process. Lett.*, 110(8-9):279–281, 2010.

[HPT19] Thomas Haines, Dirk Pattinson, and Mukesh Tiwari. Verifiable Homomorphic Tallying for the Schulze Vote Counting Scheme. In *Verified Software. Theories, Tools, and Experiments - 11th International Conference, VSTTE 2019, New York City, NY, USA, July 13-14, 2019, Revised Selected Papers*, volume 12031 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2019.

[HSB21] Lucca Hirschi, Lara Schmid, and David A. Basin. Fixing the Achilles Heel of E-Voting: The Bulletin Board. In *34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia, June 21-25, 2021*, pages 1–17. IEEE, 2021.

[IBM20] IBM. IBM Hyper Protect Virtual Servers. `https://www.ibm.com/products/hyper-protect-virtual-servers` (accessed 19.06.2023), 2020.

[Ind19] Forbes India. Forbes india investigation: India's most gerrymandered constituencies. `https://www.forbesindia.com/article/special/forbes-india-investigation-indias-most-gerrymandered-constituencies/53011/1` (accessed 19.06.2023), 2019.

[Ins23] National Democratic Institute. Internet voting in estonia. `https://www.ndi.org/e-voting-guide/examples/internet-voting-in-estonia` (accessed 19.06.2023), 2023.

[ISW21] Yuval Ishai, Hang Su, and David J. Wu. Shorter and Faster Post-Quantum Designated-Verifier zkSNARKs from Lattices. In *CCS '21: 2021 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, Republic of Korea, November 15 - 19, 2021*, pages 212–234. ACM, 2021.

[JCJ05] A. Juels, D. Catalano, and M. Jakobsson. Coercion-resistant Electronic Elections. In *Proceedings of Workshop on Privacy in the Eletronic Society (WPES 2005)*, pages 61–70. ACM Press, 2005.

[JRRS19] Wojciech Jamroga, Peter B. Rønne, Peter Y. A. Ryan, and Philip B. Stark. Risk-Limiting Tallies. In *Electronic Voting - 4th International Joint Conference, E-Vote-ID 2019, Bregenz, Austria, October 1-4, 2019, Proceedings*, volume 11759 of *Lecture Notes in Computer Science*, pages 183–199. Springer, 2019.

[Kam22] Ram Kumar Kamat. Sainte-laguë method to decide pr seats. `https://thehimalayantimes.com/nepal/sainte-lague-method-to-decide-pr-seats` (accessed 19.06.2023), 2022.

[KKL+18] Aggelos Kiayias, Annabell Kuldmaa, Helger Lipmaa, Janno Siim, and Thomas Zacharias. On the Security Properties of e-Voting Bulletin Boards. In *SCN 2018, Proceedings*, pages 505–523, 2018.

[KLM+20a] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *IEEE European Symposium on Security and Privacy, EuroS&P 2020, Genoa, Italy, September 7-11, 2020*, pages 216–235. IEEE, 2020.

[KLM+20b] Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. Technical Report 2020/405, Cryptology ePrint Archive, 2020.

[KMST16] Ralf Küsters, Johannes Müller, Enrico Scapin, and Tomasz Truderung. sElect: A Lightweight Verifiable Remote Voting System. In *IEEE 29th Computer Security Foundations Symposium (CSF 2016)*, pages 341–354. IEEE Computer Society, 2016.

[KMW12] Shahram Khazaei, Tal Moran, and Douglas Wikström. A Mix-Net from Any CCA2 Secure Cryptosystem. In Xiaoyun Wang and Kazue Sako, editors, *Advances in Cryptology - ASIACRYPT 2012 - 18th International Conference on the Theory and Application of Cryptology and Information Security, Proceedings*, volume 7658 of *Lecture Notes in Computer Science*, pages 607–625. Springer, 2012.

[KOKV11] Fatih Karayumak, Maina M. Olembo, Michaela Kauer, and Melanie Volkamer. Usability Analysis of Helios - An Open Source Verifiable Remote Electronic Voting System. In Hovav Shacham and Vanessa Teague, editors, *2011 Electronic Voting Technology Workshop / Workshop on Trustworthy Elections, EVT/WOTE '11*. USENIX Association, 2011.

[KTV10] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Accountability: definition and relationship to verifiability. In *Proceedings of the 17th ACM Conference on Computer and Communications Security, CCS 2010, Chicago, Illinois, USA, October 4-8, 2010*, pages 526–535. ACM, 2010.

[KTV11] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Verifiability, Privacy, and Coercion-Resistance: New Insights from a Case Study. In *32nd IEEE Symposium on Security and Privacy, SP 2011, 22-25 May 2011, Berkeley, California, USA*, pages 538–553. IEEE Computer Society, 2011.

[KTV12] Ralf Küsters, Tomasz Truderung, and Andreas Vogt. Clash Attacks on the Verifiability of E-Voting Systems. In *33rd IEEE Symposium on Security and Privacy (S&P 2012)*, pages 395–409. IEEE Computer Society, 2012.

[KTV+19] Ralf Küsters, Tomasz Truderung, Melanie Volkamer, Bernhard Beckert, Achim Brelle, Rüdiger Grimm, Nicolas Huber, Michael Kirsten, Jörn Müller-Quade, Maximilian Noppel, Kai Reinhard, Jonas Schwab, Rebecca Schwerdt, and Cornelia Winter. GI Elections with POLYAS: a Road to End-to-End Verifiable Elections. In *Fourth International Joint Conference on Electronic Voting (E-Vote-ID 2019)*, 2019.

[Küs06] R. Küsters. Simulation-Based Security with Inexhaustible Interactive Turing Machines. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW-19 2006)*, pages 309–320. IEEE Computer Society, 2006.

[KZGM21] Sanket Kanjalkar, Ye Zhang, Shreyas Gandlur, and Andrew Miller. Publicly Auditable MPC-as-a-Service with succinct verification and universal setup. In *IEEE European Symposium on Security and Privacy Workshops, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 386–411. IEEE, 2021.

[KZM+15] Ahmed Kosba, Zhichao Zhao, Andrew Miller, Yi Qian, Hubert Chan, Charalampos Papamanthou, Rafael Pass, Elaine Shi, et al. CØCØ: A Framework for Building Composable Zero-Knowledge Proofs. *Cryptology ePrint Archive*, 2015.

[KZZ15a] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. DEMOS-2: Scalable E2E Verifiable Elections without Random Oracles. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 352–363. ACM, 2015.

[KZZ15b] Aggelos Kiayias, Thomas Zacharias, and Bingsheng Zhang. End-to-End Verifiable Elections in the Standard Model. In *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 468–498. Springer, 2015.

[LAA+23a] Julian Liedtke, Jan Adomat, Alexander Aßenmacher, Patrick Baisch, Linus Fischer, Jonas Geiselhart, Alex Heller, Julian Kieslinger, Mike Lauer, Paul Mayer, Xuan Viet Pham, André Sperrle, Carmen Wabartha, Pia Wippermann, and Ralf Küsters. Implementation of Ordinos., 2023. `https://github.com/JulianLiedtke/Ordinos-PhD`.

[LAA+23b] Julian Liedtke, Jan Adomat, Alexander Aßenmacher, Patrick Baisch, Linus Fischer, Jonas Geiselhart, Alex Heller, Julian Kieslinger, Mike Lauer, Paul Mayer, Xuan Viet

Pham, André Sperrle, Carmen Wabartha, Pia Wippermann, and Ralf Küsters. Ordinos: Remote Verifiable Tally-Hiding E-Voting - A Fully-Fledged Web-Based Implementation. In *Proceedings of E-Vote-ID 2023*, 2023. To appear.

[LCKO19] Jiwon Lee, Jaekyoung Choi, Jihye Kim, and Hyunok Oh. SAVER: Snark-friendly, Additively-homomorphic, and Verifiable Encryption and decryption with Rerandomization. *IACR Cryptol. ePrint Arch.*, 2019:1270, 2019.

[Lij83] Arend Lijphart. Degrees of proportionality of proportional representation formulas. *RIVISTA ITALIANA DI SCIENZA POLITICA*, 13(2):295–305, 1983.

[LNP22] Vadim Lyubashevsky, Ngoc Khanh Nguyen, and Maxime Plançon. Lattice-Based Zero-Knowledge Proofs and Applications: Shorter, Simpler, and More General. In *Advances in Cryptology - CRYPTO 2022 - 42nd Annual International Cryptology Conference, CRYPTO 2022, Santa Barbara, CA, USA, August 15-18, 2022, Proceedings, Part II*, volume 13508 of *Lecture Notes in Computer Science*, pages 71–101. Springer, 2022.

[LNPS21] Vadim Lyubashevsky, Ngoc Khanh Nguyen, Maxime Plançon, and Gregor Seiler. Shorter Lattice-Based Group Signatures via "Almost Free" Encryption and Other Optimizations. In *Advances in Cryptology - ASIACRYPT 2021 - 27th International Conference on the Theory and Application of Cryptology and Information Security, Singapore, December 6-10, 2021, Proceedings, Part IV*, volume 13093 of *Lecture Notes in Computer Science*, pages 218–248. Springer, 2021.

[LS12] Mark Lindeman and Philip B. Stark. A Gentle Introduction to Risk-Limiting Audits. *IEEE Secur. Priv.*, 10(5):42–49, 2012.

[LT13] Helger Lipmaa and Tomas Toft. Secure Equality and Greater-Than Tests with Sublinear Online Complexity. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 645–656. Springer, 2013.

[Mai20] Maine State Legislature. Ranked Choice Voting in Maine. `http://legislature.maine.gov/lawlibrary/ranked-choice-voting-in-maine/9509` (accessed 19.06.2023), 2020.

[MBKM19] Mary Maller, Sean Bowe, Markulf Kohlweiss, and Sarah Meiklejohn. Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security, CCS 2019, London, UK, November 11-15, 2019*, pages 2111–2128. ACM, 2019.

[MoJ93] New Zealand Ministry of Justice. Electoral Act 1993. https://legislation.govt.nz/act/public/1993/0087/latest/DLM307519.html (accessed 19.06.2023), 1993.

[Mon87] Peter L Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Mathematics of computation*, 48(177):243–264, 1987.

[Mül19] Johannes Müller. *Design and Cryptographic Security Analysis of E-Voting Protocols*. PhD thesis, University of Stuttgart, Germany, 2019.

[NIS23] NIST. Digital Signature Standard (DSS), 2023. https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.186-5.pdf.

[NS10] Takashi Nishide and Kouichi Sakurai. Distributed Paillier Cryptosystem without Trusted Dealer. In *Information Security Applications - 11th International Workshop, WISA 2010, Jeju Island, Korea, August 24-26, 2010, Revised Selected Papers*, volume 6513 of *Lecture Notes in Computer Science*, pages 44–60. Springer, 2010.

[NSW20] NSW Government. Constitution Act No 32. https://legislation.nsw.gov.au/~/view/act/1902/32 (accessed 19.06.2023), 2020.

[OB22] Alex Ozdemir and Dan Boneh. Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, pages 4291–4308. USENIX Association, 2022.

[Okt22] Noory Okthariza. Explaining party fragmentation at district-level indonesia. *Asian Journal of Comparative Politics*, 7(4):1008–1024, 2022.

[OS01] Katsuyuki Okeya and Kouichi Sakurai. Efficient Elliptic Curve Cryptosystems from a Scalar Multiplication Algorithm with Recovery of the y-Coordinate on a Montgomery-Form Elliptic Curve. In *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, volume 2162 of *Lecture Notes in Computer Science*, pages 126–141. Springer, 2001.

[Pai99] Pascal Paillier. Public-Key Cryptosystems Based on Composite Degree Residuosity Classes. In *Advances in Cryptology - EUROCRYPT '99, Proceeding*, pages 223–238, 1999.

[PCGK17] Jordi Puiggali, Jordi Cucurull, Sandra Guasch, and Robert Krimmer. Verifiability Experiences in Government Online Voting Systems. In *Electronic Voting - Second International Joint Conference, E-Vote-ID 2017, Bregenz, Austria, October 24-27, 2017, Proceedings*, volume 10615 of *Lecture Notes in Computer Science*, pages 248–263. Springer, 2017.

[Ped91] Torben P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Advances in Cryptology - CRYPTO '91, 11th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1991, Proceedings*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[Per20] Personal communication (email) with Philip Wright, Technical Director of CES, 2020.

[PHGR13] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *2013 IEEE Symposium on Security and Privacy, SP 2013, Berkeley, CA, USA, May 19-22, 2013*, pages 238–252. IEEE Computer Society, 2013.

[Pie10] Wolter Pieters. Verifiability of Electronic Voting: Between Confidence and Trust. In *Data Protection in a Profiled World*, pages 157–175. Springer, 2010.

[Pol23] Polyas, 2023. `https://www.polyas.com/churches/church-council-elections/case-study` (19.06.2023).

[Pro11] ProPublica. How democrats fooled california's redistricting commission. `https://www.propublica.org/article/how-democrats-fooled-californias-redistricting-commission` (accessed 19.06.2023), 2011.

[Rao10] GVLN Rao. Democracy at risk! citizens for verifiability, transparency& accountability in elections, 2010.

[RCPT19] Kim Ramchen, Chris Culnane, Olivier Pereira, and Vanessa Teague. Universally Verifiable MPC and IRV Ballot Counting. In *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 301–319. Springer, 2019.

[Rep16] Republic of Nauru. Electoral Act No. 15. `http://ronlaw.gov.nr/nauru_lpms/files/acts/d83250a1ebdc56c1701fa7aa245af5b1.pdf` (accessed 19.06.2023), 2016.

[Rii02] Riigikogu. Riigikogu Election Act, 2002. `https://www.riigiteataja.ee/en/eli/ee/Riigikogu/act/514122020002/consolide` (accessed 19.06.2023).

[RLH+23] Johannes Reinhart, Bastian Lüttig, Nicolas Huber, Julian Liedtke, and Björn Annighöfer. Verifiable Computing in Avionics for Assuring Computer-Integrity without Replication. In *Digital Avionics Systems Conference 2023, Barcelona, Spain, October 1-5, 2023*, pages 1–10. IEEE, 2023.

[Sch18] Markus Schulze. The Schulze Method of Voting. *CoRR*, 2018.

[sci17] scipr-lab. libsnark. `https://github.com/scipr-lab/libsnark` (accessed 19.06.2023), 2017.

[scy17a] scytl. Scytl and Swiss Post Online Voting Solution First and Only in Switzerland to be Certified for 50% of Voters. `https://scytl.com/news/scytl-swiss-post-online-voting-solution-first-switzerland-certified-50-voters/` (accessed 19.06.2023), 2017.

[scy17b] scytl. Scytl and Swiss Post Online Voting Solution First and Only in Switzerland to be Certified for 50% of Voters. `https://scytl.com/news/scytl-swiss-post-online-voting-solution-first-switzerland-certified-50-voters/` (accessed 19.06.2023), 2017.

[Scy23a] Scytl, 2023. `https://www.scytl.com/en/customers/` (accessed 19.06.2023).

[scy23b] scytl. Scytl: Secure Online Voting and Innovative Election Solutions. `hhttps://scytl.com/` (accessed 19.06.2023), 2023.

[Set20] Srinath T. V. Setty. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *Advances in Cryptology - CRYPTO 2020 - 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17-21, 2020, Proceedings, Part III*, volume 12172 of *Lecture Notes in Computer Science*, pages 704–737. Springer, 2020.

[SFD+14] Drew Springall, Travis Finkenauer, Zakir Durumeric, Jason Kitcat, Harri Hursti, Margaret MacAlpine, and J. Alex Halderman. Security Analysis of the Estonian Internet Voting System. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 703–715. ACM, 2014.

[Soc19] Society for Industrial and Applied Mathematics (SIAM). SIAM Announces New 2020 Leadership. `https://sinews.siam.org/Details-Page/siam-announces-new-2020-leadership-1` (accessed 19.06.2023), 2019.

[SP15] Alan Szepieniec and Bart Preneel. New Techniques for Electronic Voting. *USENIX Journal of Election Technology and Systems (JETS)*, 3(2):46 – 69, 2015.

[Sta19] State of Maine. An Act To Clarify Ranked-choice Voting Laws. `https://www.mainelegislature.org/legis/bills/getPDF.asp?paper=SP0540&item=3&snum=129` (accessed 19.06.2023), 2019.

[Sto33] Jenny Stock. The playmander: Its origins, operation and effect on south australia. *Playford's South Australia: Essays on the History of South Australia*, 68:73–90, 1933.

[SV15] Berry Schoenmakers and Meilof Veeningen. Universally Verifiable Multiparty Computation from Threshold Homomorphic Cryptosystems. In *Applied Cryptography and Network Security - 13th International Conference, ACNS 2015, New York, NY, USA, June 2-5, 2015, Revised Selected Papers*, volume 9092 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[Tea19] Olivier Pereira-Vanessa Teague. Report on the swisspost-scytl e-voting system, trusted-server version, 2019.

[The11] The National Archives. Greater London Authority Act 1999. `https://www.legislation.gov.uk/ukpga/1999/29/contents` (accessed 19.06.2023), 2011.

[The13] The Swiss Federal Chancellery. Federal Chancellery Ordinance on Electronic Voting, 2013. `https://www.fedlex.admin.ch/eli/cc/2013/859/en` (accessed 19.06.2023).

[TRT22] TRTWorld. India 'gerrymanders' kashmir region in redrawn electoral map. `https://www.trtworld.com/asia/india-gerrymanders-kashmir-region-in-redrawn-electoral-map-56906` (accessed 19.06.2023), 2022.

[UK 21] UK Parliament. Voting systems in the UK. `https://www.parliament.uk/about/how/elections-and-voting/voting-systems/` (accessed 19.06.2023), 2021.

[Uni19] United States Election Assistance Commission. Advanced Voting Solutions, Inc. (AVS). `https://www.eac.gov/voting-equipment/registered-manufacturers/advanced-voting-solutions-inc-avs` (accessed 19.06.2023), 2019.

[VAS19] Thijs Veugen, Thomas Attema, and Gabriele Spini. An implementation of the Paillier crypto system with threshold decryption without a trusted dealer. Technical Report 2019/1136, Cryptology ePrint Archive, 2019.

[Ver23] Verified Voting. AVS WINVote (and WINScan). `https://verifiedvoting.org/election-system/avs-winvote-and-winscan/` (accessed 19.06.2023), 2023.

[Vir15] Virginia Regulatory Town Hall. Election Integrity Security trumps convenience, decertify AVS WinVote. `https://townhall.virginia.gov/l/ViewComments.cfm?CommentID=39937` (accessed 19.06.2023), 2015.

[Vot07] Votersunite. Advanced Voting Solutions in the News — A Partial List of Documented Failures. `https://verifiedvoting.org/wp-content/uploads/2020/08/WINvote_Malfunctions_2003-2207.pdf` (accessed 19.06.2023), 2007.

[WB09] Roland Wen and Richard Buckland. Minimum Disclosure Counting for the Alternative Vote. In *E-Voting and Identity, Second International Conference, VoteID 2009, Luxembourg, September 7-8, 2009. Proceedings*, volume 5767 of *Lecture Notes in Computer Science*, pages 122–140. Springer, 2009.

[WK23] Jan Willemson and Kristjan Krips. Estimating Carbon Footprint of Paper and Internet Voting. In *Proceedings of E-Vote-ID 2023*, pages 140–155. Springer, 2023.

[WLH+23a] Carmen Wabartha, Julian Liedtke, Nicolas Huber, Daniel Rausch, and Ralf Küsters. Fully Tally-Hiding Verifiable E-Voting for Real-World Elections with Seat-Allocations. In *28th European Symposium on Research in Computer Security (ESORICS 2023)*, volume 14346 of *Lecture Notes in Computer Science*, pages 209–228. Springer, 2023.

[WLH+23b] Carmen Wabartha, Julian Liedtke, Nicolas Huber, Daniel Rausch, and Ralf Küsters. Fully Tally-Hiding Verifiable E-Voting for Real-World Elections with Seat-Allocations. Technical Report 2023/1289, Cryptology ePrint Archive, 2023.

[WWH+10] Scott Wolchok, Eric Wustrow, J. Alex Halderman, Hari K. Prasad, Arun Kankipati, Sai Krishna Sakhamuri, Vasavya Yagati, and Rop Gonggrijp. Security analysis of India's electronic voting machines. In *ACM Conference on Computer and Communications Security (CCS 2010)*, pages 1–14, 2010.

[Zha23] Zhenfei Zhang. Origami: Fold a Plonk for Ethereum's VDF. Technical Report 2023/384, Cryptology ePrint Archive, 2023.

Academic Curriculum and Publications

Academic Curriculum

April 2018 – February 2024	**University of Stuttgart, Germany** Ph.D. student at the Institute of Information Security Supervisor: Prof. Dr. Ralf Küsters
October 2016 – March 2018	**University of Stuttgart, Germany** Master of Science in Computer Science Thesis title: *Nicht-interaktive Zero-Knowledge Beweise von Wissen mittels Fiat-Shamir Transformation* Supervisor: Prof. Dr. Ralf Küsters
October 2013 – September 2016	**University of Stuttgart, Germany** Bachelor of Science in Computer Science Thesis title: *Concept Drift and Adaptation for Emotion Detection in Twitter* Supervisor: Prof. Dr. Sebastian Padó

List of Publications

- Ralf Küsters, Julian Liedtke, Johannes Müller, Daniel Rausch, and Andreas Vogt. Ordinos: A Verifiable Tally-Hiding E-Voting System. In *IEEE European Symposium on Security and Privacy (EuroS&P) 2020*.
- Fabian Hertel, Nicolas Huber, Jonas Kittelberger, Ralf Küsters, Julian Liedtke and Daniel Rausch. Extending Ordinos: First Tally-Hiding Implementation of Borda, Condorcet and IRV. In *International Joint Conference on Electronic Voting (E-Vote-ID) 2021*.
- Nicolas Huber, Ralf Küsters, Toomas Krips, Julian Liedtke, Johannes Müller, Daniel Rausch, Pascal Reisert, and Andreas Vogt. Kryvos: Publicly Tally-Hiding Verifiable E-Voting. In *ACM Conference on Computer and Communications Security (CCS) 2022*.
- Carmen Wabartha, Julian Liedtke, Nicolas Huber, Daniel Rausch, Ralf Küsters. Fully Tally-Hiding Verifiable E-Voting for Real-World Elections with Seat-Allocations. In *European Symposium on Research in Computer Security (ESORICS) 2023*.
- Julian Liedtke, Jan Adomat, Alexander Aßenmacher, Patrick Baisch, Linus Fischer, Jonas Geiselhart, Alex Heller, Julian Kieslinger, Mike Lauer, Paul Mayer, Xuan Viet Pham, André

Sperrle, Carmen Wabartha, Pia Wippermann, and Ralf Küsters. Ordinos: Remote Verifiable Tally-Hiding E-Voting - A Fully-Fledged Web-Based Implementation. In *International Joint Conference on Electronic Voting (E-Vote-ID) 2023*.

– Reinhart, Johannes and Lüttig, Bastian and Huber, Nicolas and Liedtke, Julian and Annighöfer, Björn. Verifiable Computing in Avionics for Assuring Computer-Integrity without Replication In *Digital Avionics Systems Conference (DASC) 2023*.